305-CD-042-001

EOSDIS Core System Project

Flight Operations Segment (FOS) Command Management Design Specification for the ECS Project

October 1995

Hughes Information Technology Corporation
Upper Marlboro, MD

Flight Operation Segment (FOS) Command Management Design Specification for the ECS Project

October 1995

Prepared Under Contract NAS5-60000
CDRL Item #046

APPROVED BY

Cal E. Moore, Jr. /s/ 9/29/95
Cal Moore, FOS CCB ChairmanDate
EOSDIS Core System Project

Hughes Information Technology Corporation
Upper Marlboro, Maryland

305-CD-042-001

This page intentionally left blank.

Preface

This document, one of nineteen, comprises the detailed design specification of the FOS subsystems for Releases A and B of the ECS project. This includes the FOS design to support the AM-1 launch.

The FOS subsystem design specification documents for Releases A and B of the ECS project include:

305-CD-040	FOS Design Specification (Segment Level Design)
305-CD-041	Planning and Scheduling Design Specification
305-CD-042	Command Management Design Specification
305-CD-043	Resource Management Design Specification
305-CD-044	Telemetry Design Specification
305-CD-045	Command Design Specification
305-CD-046	Real-Time Contact Management Design Specification
305-CD-047	Analysis Design Specification
305-CD-048	User Interface Design Specification
305-CD-049	Data Management Design Specification
305-CD-050	Planning and Scheduling Program Design Language (PDL)
305-CD-051	Command Management PDL
305-CD-052	Resource Management PDL
305-CD-053	Telemetry PDL
305-CD-054	Real-Time Contact Management PDL
305-CD-055	Analysis PDL
305-CD-056	User Interface PDL
305-CD-057	Data Management PDL
305-CD-058	Command PDL

Object models presented in this document have been exported directly from CASE tools and in some cases contain too much detail to be easily readable within hard copy page constraints. The reader is encouraged to view these drawings on line using the Portable Document Format (PDF) electronic copy available via the ECS Data Handling System (EDHS) at URL http://edhs1.gsfc.nasa.gov.

Abstract

The FOS Design Specification consists of a set of 19 documents that define the FOS detailed design. The first document, the FOS Segment Level Design, provides an overview of the FOS segment design, the architecture, and analyses and trades. The next nine documents provide the detailed design for each of the nine FOS subsystems. The last nine documents provide the PDL for the nine FOS subsystems. It also allocates the level 4 FOS requirements to the subsystem design.

Keywords: FOS, design, specification, analysis, IST, EOC

This page intentionally left blank.

Change Information Page

List of Effective Pages	
Page Number	**Issue**
Title	Original
iii through xii	Original
1 -1 and 1-2	Original
2-1 through 2-4	Original
3-1 through 3-230	Original
AB-1 through AB-8	Original
GL-1 through GL-8	Original

Document History			
Document Number	**Status/Issue**	**Publication Date**	**CCR Number**
305-CD-042-001	Original	October 1995	95-0671

This page intentionally left blank.

Contents

Figures

Tables

Abbreviations and Acronyms

Glossary

1. Introduction

1.1 Identification

The contents of this document defines the design specification for the Flight Operations Segment (FOS). Thus, this document addresses the Data Item Description (DID) for CDRL Item 046 305/DV2 under Contract NAS5-60000.

1.2 Scope

The Flight Operations Segment (FOS) Design Specification defines the detailed design of the FOS. It allocates the Level 4 FOS requirements to the subsystem design. It also defines the FOS architectural design. In particular, this document addresses the Data Item Description (DID) for CDRL # 046, the Segment Design Specification.

This document reflects the August 23, 1995 Technical Baseline maintained by the contractor configuration control board in accordance with ECS Technical Direction No. 11, dated December 6, 1994. It covers releases A and B for FOS. This corresponds to the design to support the AM-1 launch.

1.3 Purpose

The FOS Design Specification consists of a set of 19 documents that define the FOS detailed design. The first document, the FOS Segment Level Design, provides an overview of the FOS segment design, the architecture, and analyses and trades. The next nine documents provide the detailed design for each of the nine FOS subsystems. The last nine documents provide the PDL for the nine FOS subsystems.

1.4 Status and Schedule

This submittal of DID 305/DV2 incorporates the FOS detailed design performed during the Critical Design Review (CDR) time frame. This document is under the ECS Project configuration control.

1.5 Document Organization

305-CD-040 contains the overview, the FOS segment models, the FOS architecture, and FOS analyses and trades performed during the design phase.

305-CD-041 contains the detailed design for Planning and Scheduling Design Specification.

305-CD-042 contains the detailed design for Command Management Design Specification.

305-CD-043 contains the detailed design for Resource Management Design Specification.

305-CD-044 contains the detailed design for Telemetry Design Specification.

305-CD-045 contains the detailed design for Command Design Specification.

305-CD-046 contains the detailed design for Real-Time Contact Management Design Specification.

305-CD-047 contains the detailed design for Analysis Design Specification.

305-CD-048 contains the detailed design for User Interface Design Specification.

305-CD-049 contains the detailed design for Data Management Design Specification.

305-CD-050 contains Planning and Scheduling PDL.

305-CD-051 contains Command Management PDL.

305-CD-052 contains Resource Management PDL.

305-CD-053 contains the Telemetry PDL.

305-CD-054 contains the Real-Time Contact Management PDL.

305-CD-055 contains the Analysis PDL.

305-CD-056 contains the User Interface PDL.

305-CD-057 contains the Data Management PDL.

305-CD-058 contains the Command PDL.

Appendix A of the first document contains the traceability between Level 4 Requirements and the design. The traceability maps the Level 4 requirements to the objects included in the subsystem object models.

Glossary contains the key terms that are included within this design specification.

Abbreviations and acronyms contains an alphabetized list of the definitions for abbreviations and acronyms used within this design specification.

2. Related Documentation

2.1 Parent Document

The parent documents are the documents from which this FOS Design Specification's scope and content are derived.

194-207-SE1-001	System Design Specification for the ECS Project
304-CD-001-002	Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 1: General Requirements
304-CD-004-002	Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 2: AM-1 Mission Specific

2.2 Applicable Documents

The following documents are referenced within this FOS Design Specification or are directly applicable, or contain policies or other directive matters that are binding upon the content of this volume.

194-219-SE1-020	Interface Requirements Document Between EOSDIS Core System (ECS) and NASA Institutional Support Systems
209-CD-002-002	Interface Control Document Between EOSDIS Core System (ECS) and ASTER Ground Data System, Preliminary
209-CD-003-002	Interface Control Document Between EOSDIS Core System (ECS) and the EOS-AM Project for AM-1 Spacecraft Analysis Software, Preliminary
209-CD-004-002	Data Format Control Document for the Earth Observing System (EOS) AM-1 Project Data Base, Preliminary
209-CD-025-001	ICD Between ECS and AM1 Project Spacecraft Software Development and Validation Facilities (SDVF)
311-CD-001-003	Flight Operations Segment (FOS) Database Design and Database Schema for the ECS Project
502-ICD-JPL/GSFC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Jet Propulsion Laboratory and the Goddard Space Flight Center for GSFC Missions Using the Deep Space Network
530-ICD-NCCDS/MOC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Goddard Space Flight Center Mission Operations Centers and the Network Control Center Data System
530-ICD-NCCDS/POCC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Goddard Space Flight Center Payload Operations Control Centers and the Network Control Center Data System
530-DFCD-NCCDS/POCC	Goddard Space Flight Center/MO&DSD, Data Format control Doc-

	ument Between the Goddard Space Flight Center Payload Operations Control Centers and the Network Control Center Data System
540-041	Interface Control Document (ICD) Between the Earth Observing System (EOS) Communications (Ecom) and the EOS Operations Center (EOC), Review
560-EDOS-0230.0001	Goddard Space Flight Center/MO&DSD, Earth Observing System (EOS) Data and Operations System (EDOS) Data Format Requirements Document (DFRD)
ICD-106	Martin Marietta Corporation, Interface Control Document (ICD) Data Format Control Book for EOS-AM Spacecraft
none	Goddard Space Flight Center, Earth Observing System (EOS) AM-1 Flight Dynamics Facility (FDF) / EOS Operations Center (EOC) Interface Control Document

2.3 Information Documents

2.3.1 Information Document Referenced

The following documents are referenced herein and, amplify or clarify the information presented in this document. These documents are not binding on the content of this FOS Design Specification.

194-201-SE1-001	Systems Engineering Plan for the ECS Project
194-202-SE1-001	Standards and Procedures for the ECS Project
193-208-SE1-001	Methodology for Definition of External Interfaces for the ECS Project
308-CD-001-004	Software Development Plan for the ECS Project
194-501-PA1-001	Performance Assurance Implementation Plan for the ECS Project
194-502-PA1-001	Contractor's Practices & Procedures Referenced in the PAIP for the ECS Project
604-CD-001-004	Operations Concept for the ECS Project: Part 1-- ECS Overview, 6/95
604-CD-002-001	Operations Concept for the ECS project: Part 2B -- ECS Release B, Annotated Outline, 3/95
604-CD-003-001	ECS Operations Concept for the ECS Project: Part 2A -- ECS Release A, Final, 7/95
194-WP-912-001	EOC/ICC Trade Study Report for the ECS Project, Working Paper
194-WP-913-003	User Environment Definition for the ECS Project, Working Paper
194-WP-920-001	An Evaluation of OASIS-CC for Use in the FOS, Working Paper
194-TP-285-001	ECS Glossary of Terms
222-TP-003-006	Release Plan Content Description
none	Hughes Information Technology Company, Technical Proposal for the EOSDIS Core System (ECS), Best and Final Offer

560-EDOS-0211.0001	Goddard Space Flight Center, Interface Requirements Document (IRD) Between the Earth Observing System (EOS) Data and Operations System (EDOS), and the EOS Ground System (EGS) Elements, Preliminary
NHB 2410.9A	NASA Hand Book: Security, Logistics and Industry Relations Division, NASA Security Office: Automated Information Security Handbook

This page intentionally left blank.

3. Command Management

The Command Management subsystem (CMS) is responsible for providing tools used to manage the planned operations of the EOS spacecraft and their instruments. Planned operations are managed by means of ground scripts, preplanned command procedures, and spacecraft and instrument loads containing stored commands, data, or software. Ground scripts are created by CMS based on the Detailed Activity Schedule (DAS) created by Planning & Scheduling subsystem (PAS). Preplanned command procedures are created by the Procedure Builder, described in Book 9 of the FOS Design Specification, and validated by CMS.

Loads that are ready for uplink are generated by CMS from load content information. The five types of load contents processed by CMS are: absolute time command (ATC), which are created by CMS based on the DAS from PAS; relative time sequence (RTS), which are created either by the FOS User Interface subsystem (FUI) or externally to the FOS and must follow a format defined in the PDB; table, which are created either by FUI or externally to the FOS and must follow a format defined in the PDB; microprocessor, which are created externally to the FOS; and flight software, which are created externally to the FOS. Each type of load contents is processed differently by CMS: ATC and RTS load contents consist of commands and time tags that are converted to binary; table load contents consist of data fields that are converted to binary; microprocessor and flight software load contents are already in binary form when received by CMS. Once the load contents is in binary, CMS formats the load data for uplink by the FOS Command subsystem.

The CMS also generates reports on load contents and current uplink status and maintains information on the current state of spacecraft memory.

The CMS design includes five components: Schedule Controller, which is responsible for receiving the DAS from PAS and initiating the generation of products based on the DAS; Ground Schedule, which is responsible for maintaining a continuous schedule of commands and generating products based on that schedule; Command Model, which is responsible for performing rule based constraint checks; Spacecraft Model, which models spacecraft memory; and Load Catalog, which generates and maintains loads that are available for uplink.

3.1 Command Management Context

The CMS interfaces with the other FOS subsystems and with external entities. These interfaces are shown in Figure 3.1-1 and described below.

Planning and Scheduling Subsystem - The Planning and Scheduling subsystem (PAS) generates an integrated, conflict-free schedule of space and ground activities for each spacecraft for each target day. PAS sends this schedule with a request to generate ATC loads and ground scripts based on the schedule. CMS performs command-level constraint checking and returns conflict information to PAS. PAS allows the user to override soft constraint violations and returns a constraint override status to CMS. Once the ATC load is built, CMS sends an uplink request for the load to PAS.

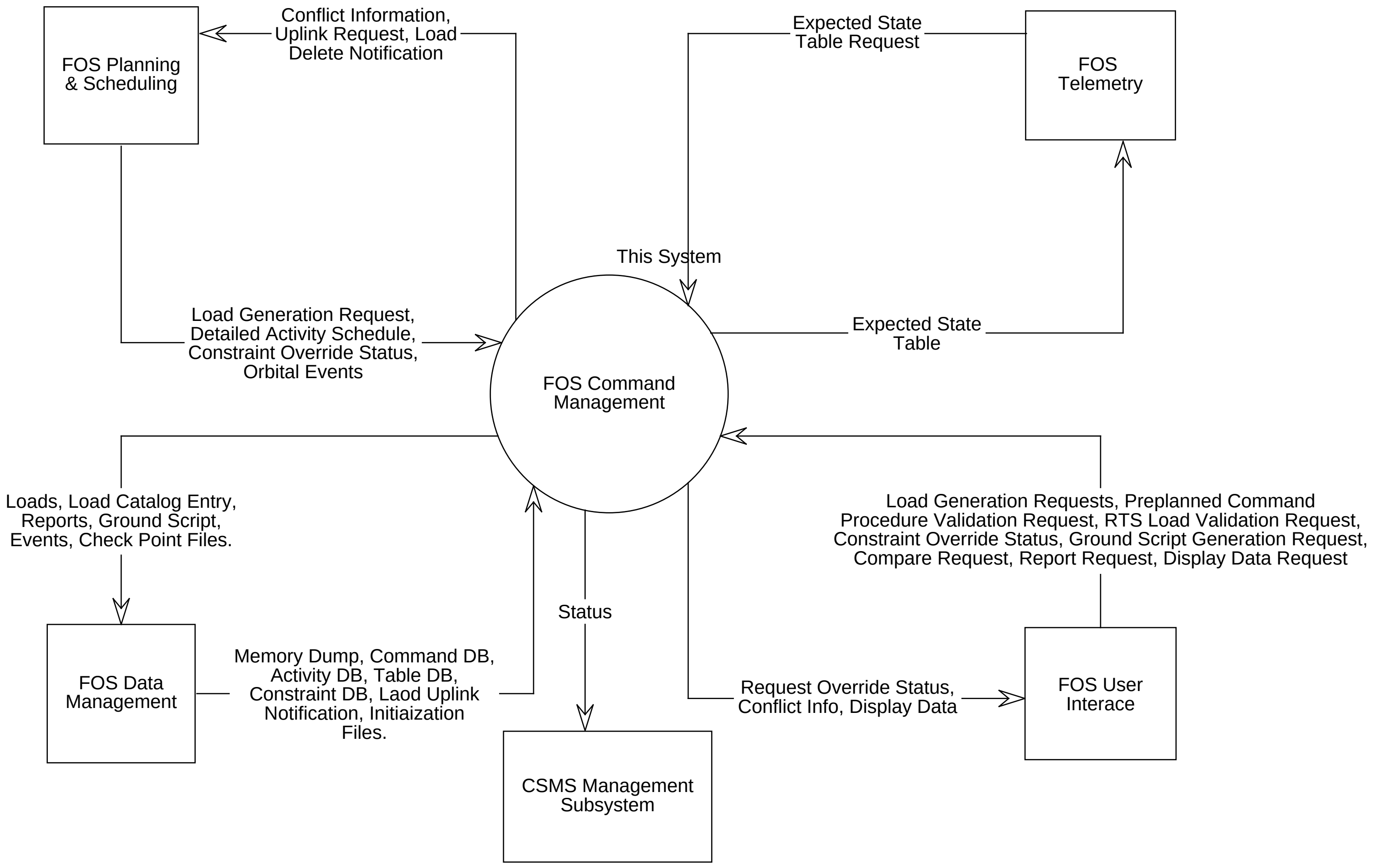

FOS Planning & Scheduling
Conflict Information, Uplink Request, Load Delete Notification
Expected State Table Request
FOS Telemetry
This System
Load Generation Request, Detailed Activity Schedule, Constraint Override Status, Orbital Events
Expected State Table
FOS Command Management
Loads, Load Catalog Entry, Reports, Ground Script, Events, Check Point Files.
Load Generation Requests, Preplanned Command Procedure Validation Request, RTS Load Validation Request, Constraint Override Status, Ground Script Generation Request, Compare Request, Report Request, Display Data Request
Status
FOS Data Management
Memory Dump, Command DB, Activity DB, Table DB, Constraint DB, Laod Uplink Notification, Initiaization Files.
Request Override Status, Conflict Info, Display Data
FOS User Interace
CSMS Management Subsystem

Figure 3.1-1. CMS Context Diagram

The schedule that PAS sends to CMS includes the orbital events that were used in generating the schedule. CMS includes these orbital events in the Integrated Report and in the ground script as comments.

CMS is also responsible for notifying PAS if a load that has been scheduled for uplink has been deleted from the load catalog.

User Interface Subsystem - The FOT requests generation of RTS, table, microprocessor, and flight software loads from load contents files via the FOS User Interface Subsystem (FUI). CMS generates the load and returns a status to FUI.

FUI also sends requests for ground script generation to CMS. CMS generates a ground script and returns a request status to FUI. CMS also provides ground script information for a report on request.

FUI sends command procedures and RTS load contents to CMS for validation. The CMS performs a constraint check of the command procedure or RTS and returns the results. FUI allows the user to override soft constraint violations and, for RTS loads, forwards the override status to CMS.

FUI sends requests for generation of a memory image from collected dump telemetry to CMS. These requests may include a request for comparison of the dump image to a load or ground reference image, or for an image report on the dump image. CMS generates the dump image and performs the comparison or generates the report as requested. A status is returned to FUI.

The CMS is also responsible for sending requested spacecraft memory image and map information to FUI for displays and reports.

Analysis Subsystem - The FOS Analysis subsystem requests generation of a table load containing spacecraft clock correlation data. CMS generates the table load and returns a status to Analysis.

Telemetry Subsystem - The FOS Telemetry subsystem requests generation of an Expected State Table by CMS. CMS generates the table and returns it to Telemetry.

Data Management Subsystem - The CMS gets definitions of spacecraft tables, activities, and commands from the Data Management Subsystem (DMS). These definitions are used in validating and generating loads. DMS also provides constraint definitions, which are used in validating loads, activities, and procedures, and command execution verification definitions, which are used in generating expected state tables.

The CMS generates loads which will be uplinked by the Command Subsystem. The CMS stores these loads, in the form of the binary uplink load, the load image, the load report, and the load contents file from which the load was generated with DMS. For each load generated, CMS adds an entry to the load catalog which is maintained by DMS.

The CMS gets raw memory dump telemetry from DMS and uses these data to construct a dump image. This dump image may be input to a comparison, input to a report, or stored with DMS for forwarding to the SCF or SDF.

Persistent data that must be maintained by CMS are periodically checkpointed to DMS and are read in when CMS is initialized. These include spacecraft memory images and maps, the ground schedule, and expected state tables.

The CMS sends event messages generated as a result of CMS processing to DMS. DMS passes load uplink completion notification messages to CMS.

The CMS stores the reports that it generates with DMS.

3.2 CMS Schedule Controller

The Schedule Controller is a persistent process that runs on the FOS Data Server. It is responsible for initiating the generation of products that are based on schedules received from PAS. These products include ATC loads, ground scripts, integrated reports, expected state tables, and command-level constraint check results.

The schedules generated by PAS are sent to Schedule Controller in the form of activity lists. Schedule Controller expands each activity in the list using a database definition of the activity and creates a directive list representing the expanded activity list. Subsequent CMS processing of the directive list by other CMS components is initiated and controlled by Schedule Controller.

3.2.1 CMS Schedule Controller Context

The CMS Schedule interfaces with several FOS subsystems, as shown in the Context Diagram and summarized below.

Planning & Scheduling:

- Sends a DAS, which is a conflict free schedule of activities, to request ATC load generation.
- Sends a Late Change, which is another form of DAS, to request regeneration of ATC load(s).
- Sends a "What if" list, which is another form of a DAS that is only constraint checked and no load is generated.
- Sends an uplink schedule, which is a DAS consisting of uplink activities. These activities are added to the ground schedule.
- Receives a CMS Status, which returns the status of DAS, Late Change or "What if" processing.
- Receives a list of uplink requests, which was generated when processing the input DAS or Late Change activity list

CMS Load Catalog:

- Receives a load deletion request, which is a DAS id for which all associated loads are to be deleted
- Receives a generate ATC load request, which is a list of directives from which an ATC load is to be built.

CMS Ground Schedule:

- Provides a list of directives which will be used for continuity information in constraint checking a directive list

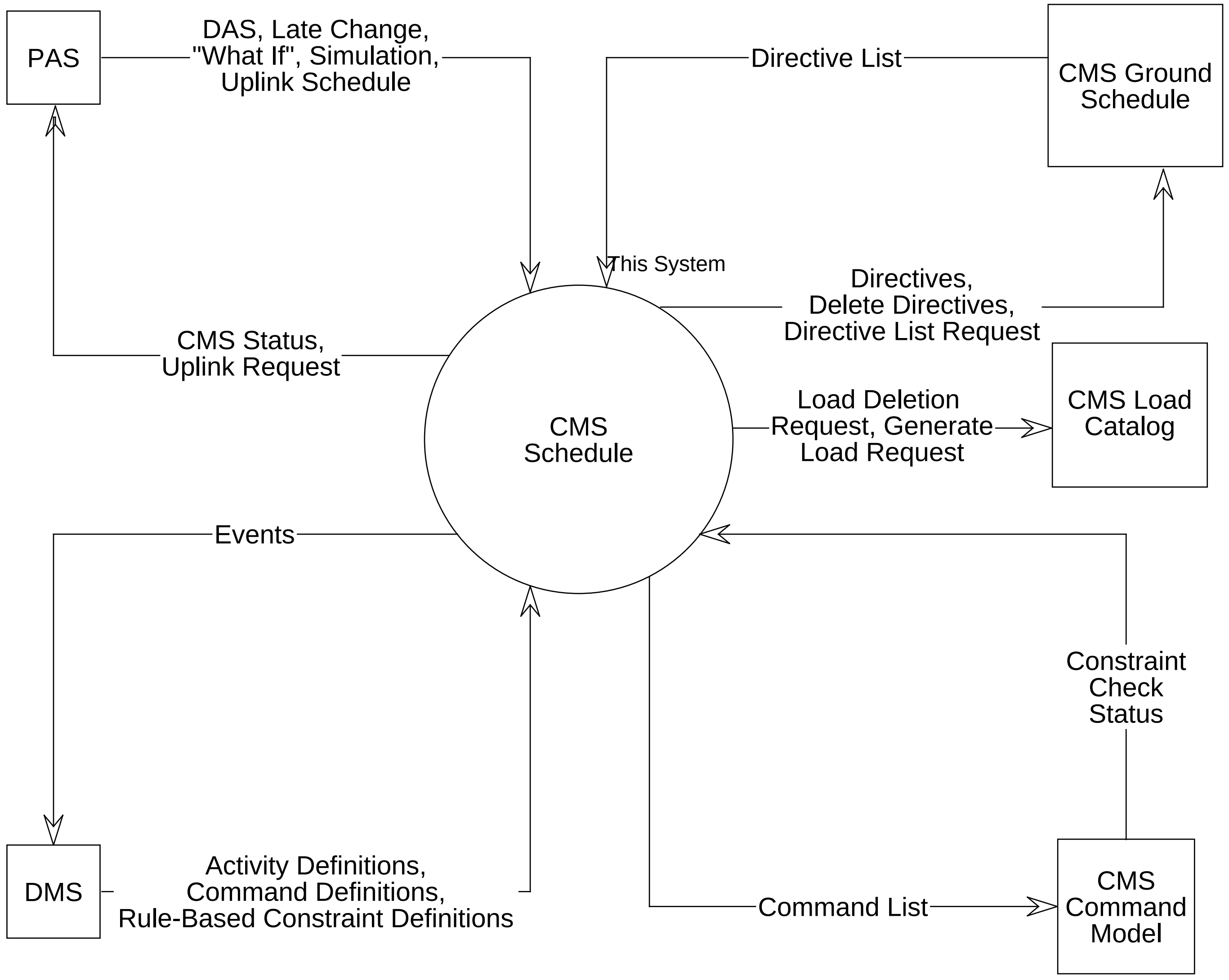

PAS
DAS, Late Change, "What If", Simulation, Uplink Schedule
Directive List
CMS Ground Schedule
This System
Directives, Delete Directives, Directive List Request
CMS Status, Uplink Request
CMS Schedule
Load Deletion Request, Generate Load Request
CMS Load Catalog
Events
Constraint Check Status
DMS
Activity Definitions, Command Definitions, Rule-Based Constraint Definitions
Command List
CMS Command Model

Figure 3.2-1. Schedule Controller Context Diagram

- Receives a delete directive list, which is a list of directives to be removed from the ground schedule.
- Receives an add directive list, which is a list of directives to be added to the ground schedule

Data Management:

- Provides activity definitions, which contain the expansion instructions for an activity.
- Provides command definitions, which include the command execution verification definitions and an optional rule based constraint definition.
- Receives events, which are status messages about CMS Schedule processing.

3.2.2 CMS Schedule Controller Interfaces

Table 3.2.2. CMS Schedule Controller Interfaces (1 of 3)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
DAS Processing	FmMsProcessSchedule	Proxy between PAS and Schedule Controller	CMS: ScheduleController	PAS: ATCLoadGenerator	1/day
	FoScDetActSched	List of scheduled activities which constitutes the Detailed Activity Schedule			
	FoScActivity	Scheduled activities that make up the Detailed Activity Schedule			
	FpCrConflictResponse	Response to soft constraints found in DAS command list			
Respond to Activity Schedule	FpCrStatusUpdater	Proxy between PAS and CMS:Schedule Controller	PAS	CMS: Schedule Controller	
	FoMsCMSStatus	Results of Activity List processing			
	FmPcUplinkSchedreq	Request for scheduling of uplink times (only for DAS and Late Change processing).			
Late Change Processing	FmMsProcessSchedule	Proxy between PAS and Schedule Controller	CMS: ScheduleController	PAS: ATCLoadGenerator	1/week
	FoScLateChange	List of scheduled activities which constitutes the Late Change Detailed Activity Schedule			
	FoScActivity	Scheduled activities that make up the Detailed Activity Schedule			
	FoMsCMSStatus	Status of Late Change processing			

Table 3.2.2. CMS Schedule Controller Interfaces (2 of 3)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
	FpCrConflictResponse	Response to soft constraints found in DAS command list			
	FmPcUplinkSchedReq	Request for scheduling of ATC load uplink			
"What If" Processing	FmMsProcessSchedule	Proxy between PAS and Schedule Controller	CMS: ScheduleController	PAS: ATCLoadGenerator	1/week
	FoScConstaintCheck	List of scheduled activities which constitutes the "What If" Activity Schedule			
	FoScActivity	Scheduled activities that make up the Activity Schedule			
	FoMsCMSStatus	Status of What If processing			
Simulation Processing	FmMsProcessSchedule	Proxy between PAS and Schedule Controller	CMS: ScheduleController	PAS: ATCLoadGenerator	1/month
	FoScSimulationSchedule	List of scheduled activities which constitutes the Simulation Activity Schedule			
	FoScActivity	Scheduled activities that make up the Activity Schedule			
	FoMsCMSStatus	Status of Simulation processing			
	FpCrConflictResponse	Response to soft constraints found in DAS command list			
Validate Commands	FmMsValidateConstraints	Proxy between CMS:CommandModel and CMS: Schedule Controller	CMS: Command Model	CMS Schedule Controller	1/day
	FmScContCk	Request for constraint check.			
	FoMsCMSStatus	Status of constraint check.			
Check for ATC Load	FmMsStoreATCLoad	Proxy between CMS:LoadCatalog and CMS: ScheduleController checking for the existence of a specified load.	CMS: Load Catalog	CMS: ScheduleController	1/week
Delete ATC Load	FmMsStoreATCLoad	Proxy between CMS:LoadCatalog and CMS: ScheduleController requesting deletion of a specified load.	CMS: Load Catalog	CMS: ScheduleController	1/month

Table 3.2.2. CMS Schedule Controller Interfaces (3 of 3)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Store ATC Load	FmMsStoreATCLoad	Proxy between CMS:LoadCatalog and CMS: ScheduleController requesting storage of an ATC load.	CMS: Load Catalog	CMS: ScheduleController	1/day
	FoLiATCLoad	Contains ATC binary load data	CMS: Load Catalog	CMS: Schedule Controller	1/day
Get ATC Buffer Start Time	FmSmMapBuffer	Proxy between CMS:Schedule Controller and CMS: Spacecraft Model.	CMS: Spacecraft Model	CMS: Schedule Controller	1/day
	FmMsATCBufferInfo	Contains Detailed Activity List and start time of ATC buffer			
Map Command into ATC Buffer	FmSmMapBuffer	Proxy Between CMS:Schedule Controller and CMS:Spacecraft Model.	CMS: SpacecraftModel	CMS: Schedule Controller	1/day
	FmMsLoadData	Contains list of directives for one load.			
	FmMsATCMapRequest	Request to Map one ATC list into the buffer model and return a list of loads.			
Delete Directives	FmGsGroundData	Proxy between CMS:Schedule Controller and CMS: Ground Schedule. Requests deletion of directives with specified DAS id(s) from the Ground Schedule	CMS: Ground Schedule	CMS: ScheduleController	1/month
Add Directives	FmGsGroundData	Proxy between CMS:Schedule Controller and CMS: Ground Schedule.	CMS: Ground Schedule	CMS: ScheduleController	1/day
	FmMnDirectiveList	List of directives to add to the Ground Schedule.			
Return Directive List	FmGsGroundData	Proxy between CMS:Schedule Controller and CMS: Ground Schedule.	CMS: Ground Schedule	CMS: ScheduleController	1/day
	FmGsListRequest	Contains list of DAS id's and a time. All directives in the Ground Schedule that occur after the time and that have DAS id's that are in the list will be returned.			
	FmMnDirectiveList	List of directives.			

3.2.3 CMS Schedule Controller Object Model

The object model for Schedule Controller is shown in Figure 3.2-2 and described below. The FmScScheduleController object receives and manages requests for schedule-related CMS services. An FoScActivityList request of type FoScDetActSched, FoScLateChange, FoScConstraintCheck, or FoScSimulationSched contains a conflict free, time ordered list of activities. The type determines if CMS will perform constraint checking only or constraint checking and ATC load generation. FmScScheduleController processes each FoScActivityList object it receives by expanding the activities in the list and building FmMnDirectiveList, which is a list of FoEcDirectives.

FmAcActivity is responsible for expanding the activity using information passed to it by FmScScheduleController and supplied by FoAcActivityDefinition to generate the absolute times and parameter list for each of the directives that make up the activity. FoAcActivityDefinition is derived from FoDbAccessor and is responsible for retrieving the database definition of an activity that FmAcActivity uses. If an activity definition includes an execute RTS or start procedure, these are expanded into a list of directives at this time.

FmScScheduleController uses the proxy FmGsGroundData to retrieve the ground and space directives from the ground schedule. The directives that are retrieved from the ground schedule will be merged with the FmMnDirectiveList to create the FmScConstCk object. This FmScConstCk object contains a time ordered list of ground directives and space directives to be constraint checked.

A message is sent to FmScCommandModel via the FmMsValidateConstraints proxy to request constraint checking the merged command list. Each command in the FmScConstCk list will be constraint checked based on an optional constraint check rule that is associated with each command. An FoMsCMSStatus reflects the status of CMS processing of an FoScActivityList and is returned to FmScScheduleController and then passed to PAS via the FpCrStatusUpdater proxy class.

FmScATCSchedule is responsible for coordinating the building of the load, generating load reports, updating the load catalog, partitioning the load, and updating the ATC buffer model. FmScATCSchedule invokes FmScMapBuffer, which is the proxy to the spacecraft model, to partition the load based on the uplink time and the available space in the ATC buffer. FmScMapBuffer returns a list of FmMsLoadData objects each which contain a load file and an associated uplink window. For each FmMsLoadData object, FmScATCSchedule creates an FoLiATCLoad object. The FoLiATCLoad object generates a binary load file and a load report. After the load is generated, FmScATCSchedule uses accesses the FmMsStoreATCLoad object. FmMsStoreATCLoad is the proxy to the load catalog and is responsible for updating the load catalog with the information in the FoLiATCLoad object. For each partition load, an FoScUplinkSchedule object, which contains the uplink window time of an ATC load, is created and sent to PAS.

FmGsGroundData is the proxy to the FmScGroundSchedule, which is responsible for adding and deleting directives in the ground schedule.

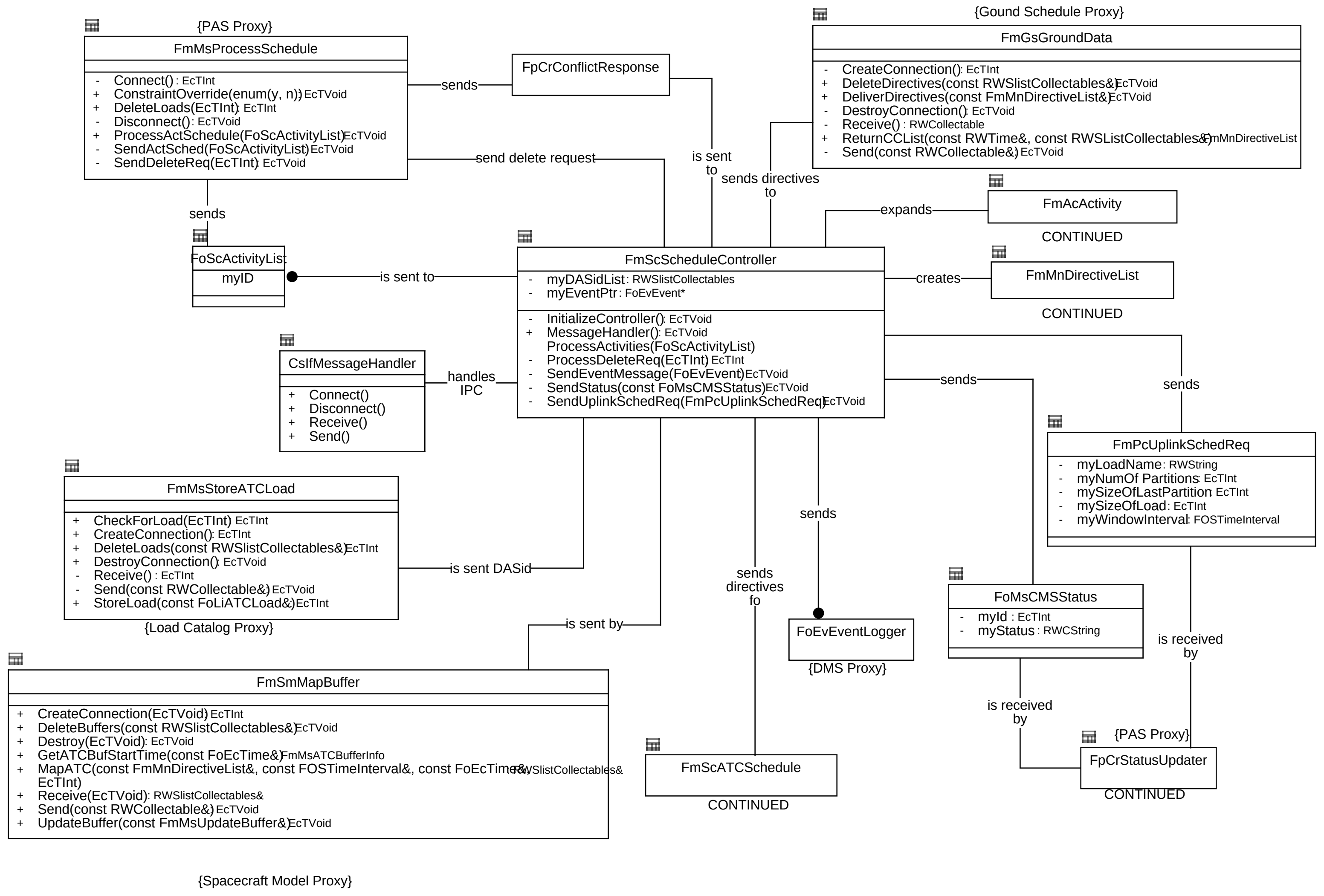

{PAS Proxy}
FmMsProcessSchedule
- Connect() : EcTInt
+ ConstraintOverride(enum(y, n)) : EcTVoid
+ DeleteLoads(EcTInt) : EcTInt
- Disconnect() : EcTVoid
+ ProcessActSchedule(FoScActivityList) : EcTVoid
- SendActSched(FoScActivityList) : EcTVoid
- SendDeleteReq(EcTInt) : EcTVoid
sends
FpCrConflictResponse
send delete request
is sent to
{Gound Schedule Proxy}
FmGsGroundData
- CreateConnection() : EcTInt
+ DeleteDirectives(const RWSlistCollectables&) : EcTVoid
+ DeliverDirectives(const FmMnDirectiveList&) : EcTVoid
- DestroyConnection() : EcTVoid
- Receive() : RWCollectable
+ ReturnCCList(const RWTime&, const RWSListCollectables&) : FmMnDirectiveList
- Send(const RWCollectable&) : EcTVoid
sends directives to
expands
FmAcActivity
CONTINUED
sends
FoScActivityList
myID
is sent to
FmScScheduleController
- myDASidList : RWSlistCollectables
- myEventPtr : FoEvEvent*
- InitializeController() : EcTVoid
+ MessageHandler() : EcTVoid
ProcessActivities(FoScActivityList)
- ProcessDeleteReq(EcTInt) : EcTInt
- SendEventMessage(FoEvEvent) : EcTVoid
- SendStatus(const FoMsCMSStatus) : EcTVoid
- SendUplinkSchedReq(FmPcUplinkSchedReq) : EcTVoid
creates
FmMnDirectiveList
CONTINUED
CsIfMessageHandler
+ Connect()
+ Disconnect()
+ Receive()
+ Send()
handles IPC
sends
sends
FmPcUplinkSchedReq
- myLoadName : RWString
- myNumOf Partitions : EcTInt
- mySizeOfLastPartition : EcTInt
- mySizeOfLoad : EcTInt
- myWindowInterval : FOSTimeInterval
FmMsStoreATCLoad
+ CheckForLoad(EcTInt) : EcTInt
+ CreateConnection() : EcTInt
+ DeleteLoads(const RWSlistCollectables&) : EcTInt
+ DestroyConnection() : EcTVoid
- Receive() : EcTInt
- Send(const RWCollectable&) : EcTVoid
+ StoreLoad(const FoLiATCLoad&) : EcTInt
{Load Catalog Proxy}
is sent DASid
sends
sends directives fo
is sent by
FoEvEventLogger
{DMS Proxy}
FoMsCMSStatus
- myId : EcTInt
- myStatus : RWCString
is received by
is received by
{PAS Proxy}
FpCrStatusUpdater
CONTINUED
FmSmMapBuffer
+ CreateConnection(EcTVoid) : EcTInt
+ DeleteBuffers(const RWslistCollectables&) : EcTVoid
+ Destroy(EcTVoid) : EcTVoid
+ GetATCBufStartTime(const FoEcTime&) : FmMsATCBufferInfo
+ MapATC(const FmMnDirectiveList&, const FOSTimeInterval&, const FoEcTime&, EcTInt) : RWSlistCollectables&
+ Receive(EcTVoid) : RWSlistCollectables&
+ Send(const RWCollectable&) : EcTVoid
+ UpdateBuffer(const FmMsUpdateBuffer&) : EcTVoid
FmScATCSchedule
CONTINUED
{Spacecraft Model Proxy}

Figure 3.2-2. Schedule Controller Object Model - Page 1

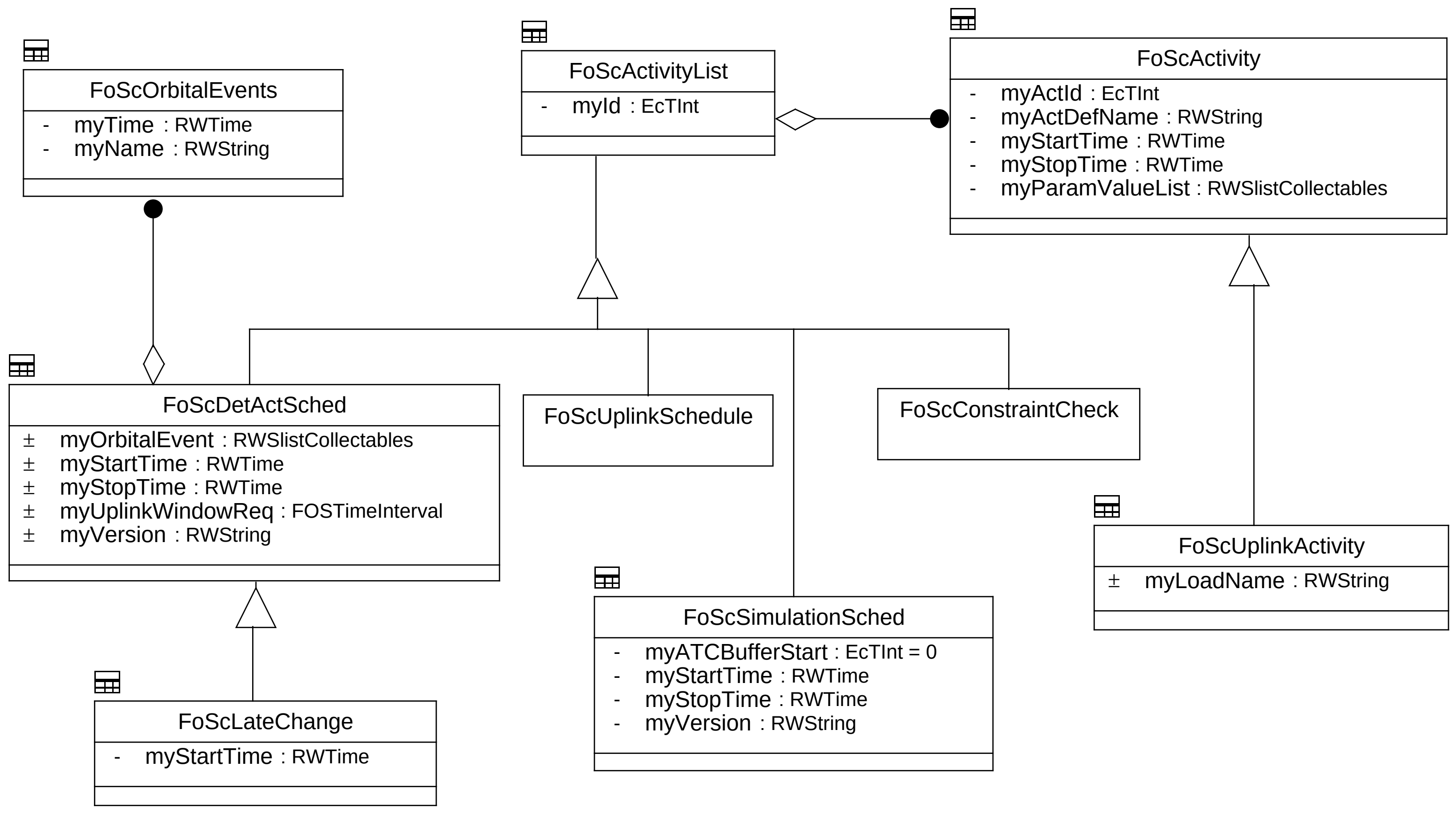
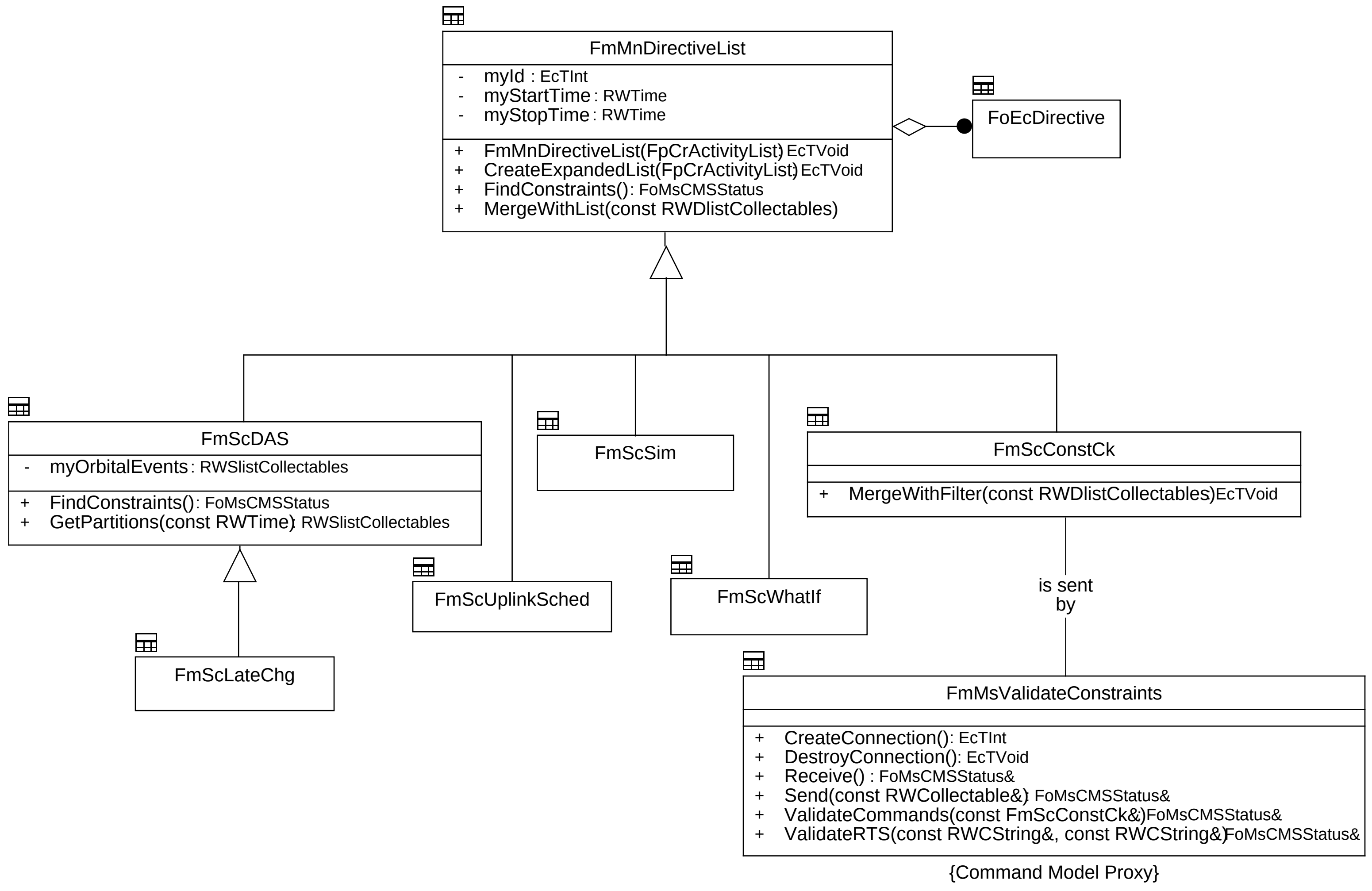
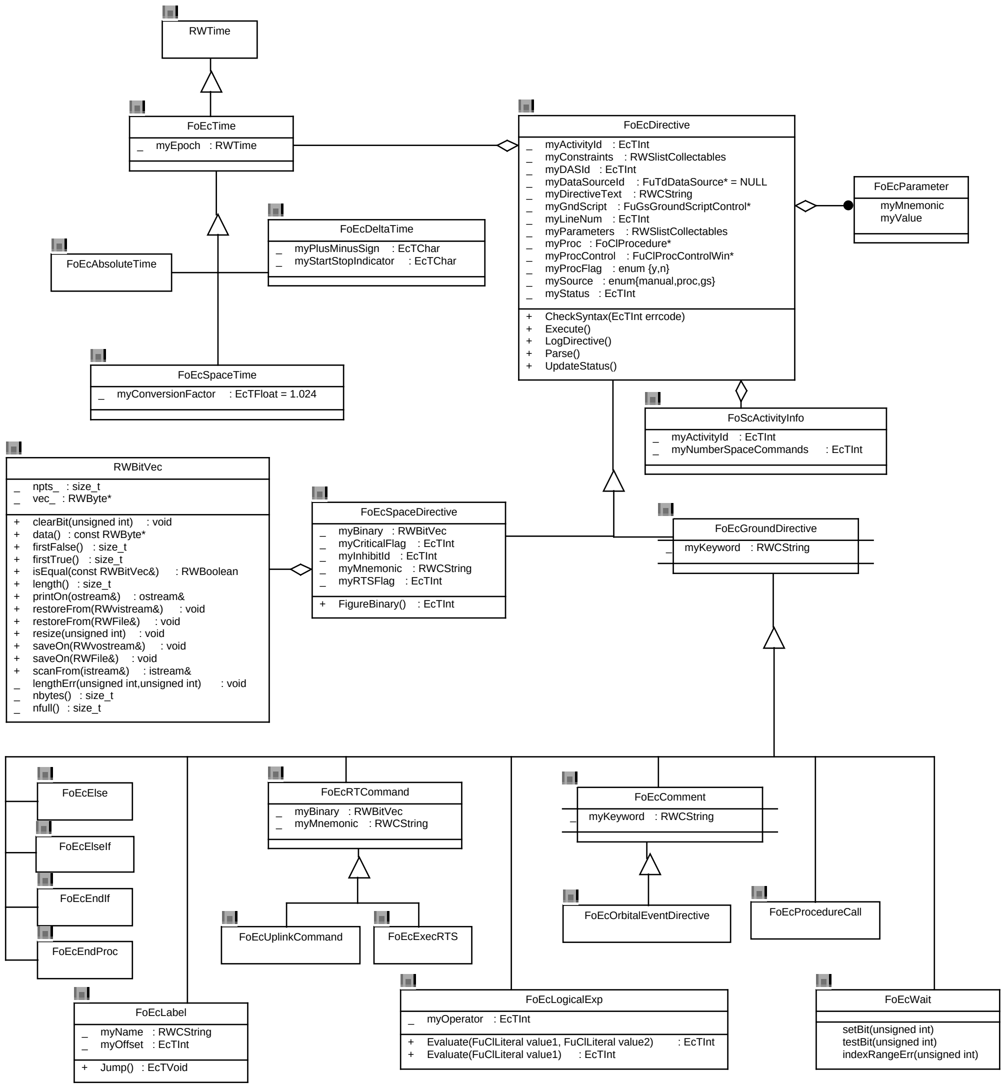


Figure 3.2-3. Schedule Controller Object Model - Page 2



FmMnDirectiveList
- myId : EcTInt
- myStartTime : RWTime
- myStopTime : RWTime
+ FmMnDirectiveList(FpCrActivityList) : EcTVoid
+ CreateExpandedList(FpCrActivityList) : EcTVoid
+ FindConstraints() : FoMsCMSStatus
+ MergeWithList(const RWDlistCollectables)
FoEcDirective
FmScDAS
- myOrbitalEvents : RWSlistCollectables
+ FindConstraints() : FoMsCMSStatus
+ GetPartitions(const RWTime) : RWSlistCollectables
FmScSim
FmScConstCk
+ MergeWithFilter(const RWDlistCollectables) EcTVoid
FmScLateChg
FmScUplinkSched
FmScWhatIf
is sent by
FmMsValidateConstraints
+ CreateConnection() : EcTInt
+ DestroyConnection() : EcTVoid
+ Receive() : FoMsCMSStatus&
+ Send(const RWCollectable&) : FoMsCMSStatus&
+ ValidateCommands(const FmScConstCk&) FoMsCMSStatus&
+ ValidateRTS(const RWCString&, const RWCString&) FoMsCMSStatus&
{Command Model Proxy}

Figure 3.2-4. Schedule Controller Object Model - Page 3



RWTime
FoEcTime
_ myEpoch : RWTime
FoEcDirective
_ myActivityId : EcTInt
_ myConstraints : RWSlistCollectables
_ myDASId : EcTInt
_ myDataSourceId : FuTdDataSource* = NULL
_ myDirectiveText : RWCString
_ myGndScript : FuGsGroundScriptControl*
_ myLineNum : EcTInt
_ myParameters : RWSlistCollectables
_ myProc : FoClProcedure*
_ myProcControl : FuClProcControlWin*
_ myProcFlag : enum {y,n}
_ mySource : enum{manual,proc,gs}
_ myStatus : EcTInt
+ CheckSyntax(EcTInt errcode)
+ Execute()
+ LogDirective()
+ Parse()
+ UpdateStatus()
FoEcParameter
myMnemonic
myValue
FoEcDeltaTime
_ myPlusMinusSign : EcTChar
_ myStartStopIndicator : EcTChar
FoEcAbsoluteTime
FoEcSpaceTime
_ myConversionFactor : EcTFloat = 1.024
FoScActivityInfo
_ myActivityId : EcTInt
_ myNumberSpaceCommands : EcTInt
RWBitVec
_ npts_ : size_t
_ vec_ : RWByte*
+ clearBit(unsigned int) : void
+ data() : const RWByte*
+ firstFalse() : size_t
+ firstTrue() : size_t
+ isEqual(const RWBitVec&) : RWBoolean
+ length() : size_t
+ printOn(ostream&) : ostream&
+ restoreFrom(RWvistream&) : void
+ restoreFrom(RWFile&) : void
+ resize(unsigned int) : void
+ saveOn(RWvostream&) : void
+ saveOn(RWFile&) : void
+ scanFrom(istream&) : istream&
_ lengthErr(unsigned int,unsigned int) : void
_ nbytes() : size_t
_ nfull() : size_t
FoEcSpaceDirective
_ myBinary : RWBitVec
_ myCriticalFlag : EcTInt
_ myInhibitId : EcTInt
_ myMnemonic : RWCString
_ myRTSFlag : EcTInt
+ FigureBinary() : EcTInt
FoEcGroundDirective
_ myKeyword : RWCString
FoEcElse
FoEcElseIf
FoEcEndIf
FoEcEndProc
FoEcLabel
_ myName : RWCString
_ myOffset : EcTInt
+ Jump() : EcTVoid
FoEcRTCommand
_ myBinary : RWBitVec
_ myMnemonic : RWCString
FoEcUplinkCommand
FoEcExecRTS
FoEcComment
_ myKeyword : RWCString
FoEcOrbitalEventDirective
FoEcProcedureCall
FoEcLogicalExp
_ myOperator : EcTInt
+ Evaluate(FuClLiteral value1, FuClLiteral value2) : EcTInt
+ Evaluate(FuClLiteral value1) : EcTInt
FoEcWait
setBit(unsigned int)
testBit(unsigned int)
indexRangeErr(unsigned int)

Figure 3.2-5. Schedule Controller Object Model - Page 4

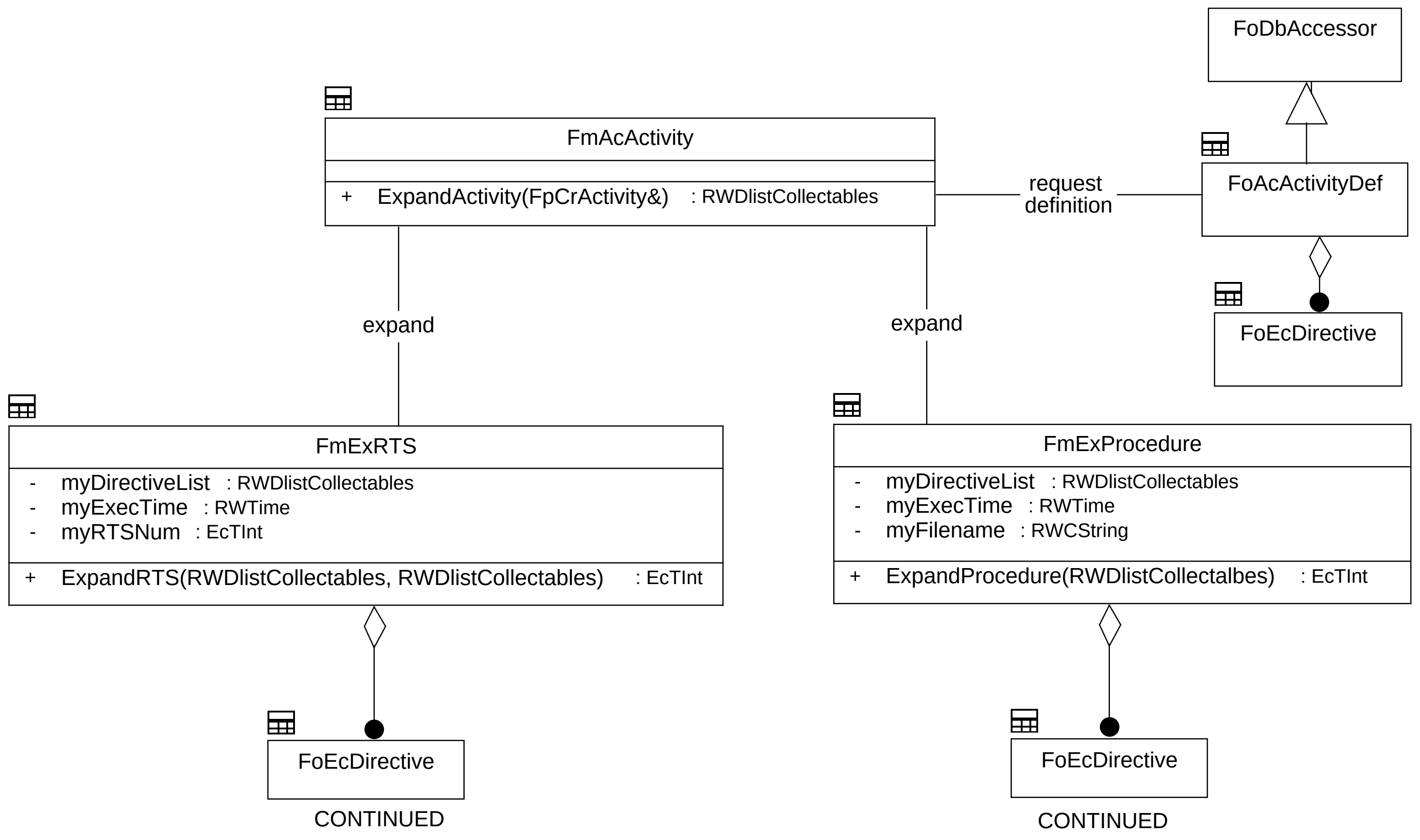

FoDbAccessor
FmAcActivity
+ ExpandActivity(FpCrActivity&) : RWDlistCollectables
request definition
FoAcActivityDef
FoEcDirective
expand
expand
FmExRTS
- myDirectiveList : RWDlistCollectables
- myExecTime : RWTime
- myRTSNum : EcTInt
+ ExpandRTS(RWDlistCollectables, RWDlistCollectables) : EcTInt
FmExProcedure
- myDirectiveList : RWDlistCollectables
- myExecTime : RWTime
- myFilename : RWCString
+ ExpandProcedure(RWDlistCollectalbes) : EcTInt
FoEcDirective
CONTINUED
FoEcDirective
CONTINUED

Figure 3.2-6. Schedule Controller Object Model - Page 5

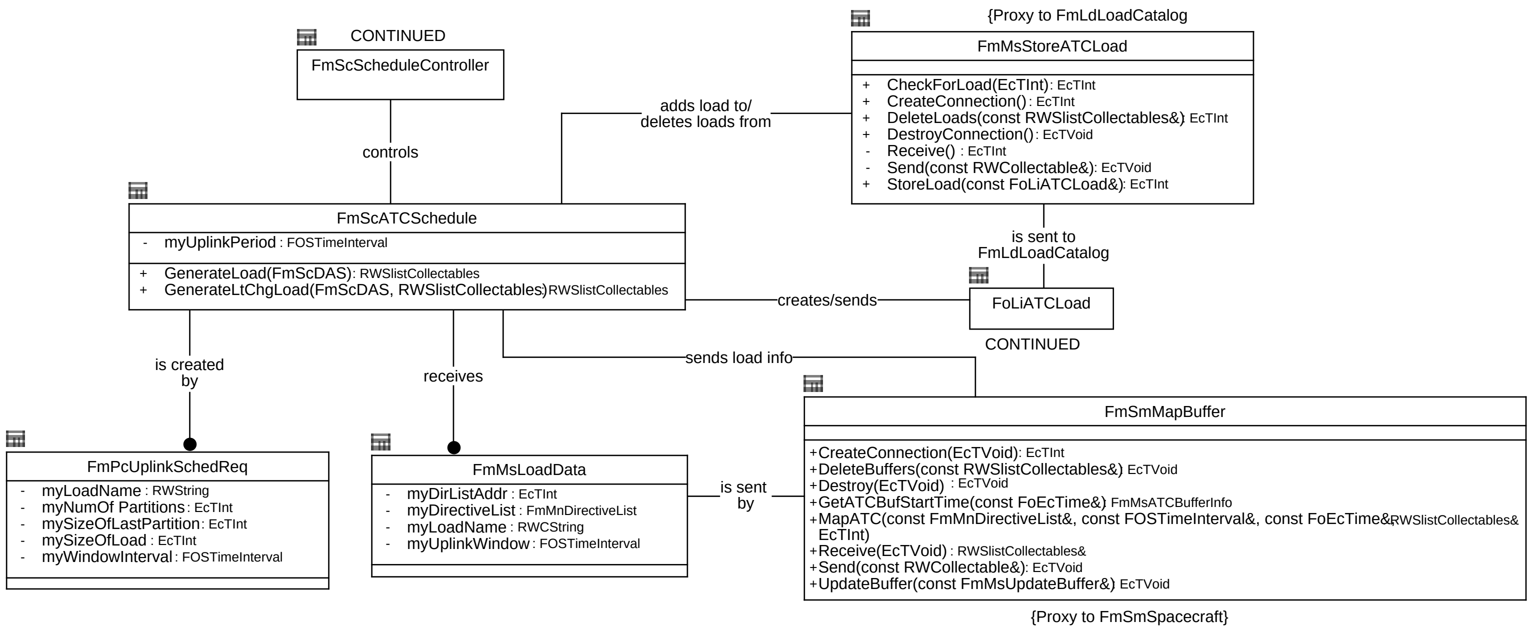

CONTINUED
FmScScheduleController
controls
adds load to/ deletes loads from
{Proxy to FmLdLoadCatalog
FmMsStoreATCLoad
+ CheckForLoad(EcTInt): EcTInt
+ CreateConnection() : EcTInt
+ DeleteLoads(const RWSlistCollectables&) EcTInt
+ DestroyConnection() : EcTVoid
- Receive() : EcTInt
- Send(const RWCollectable&): EcTVoid
+ StoreLoad(const FoLiATCLoad&): EcTInt
FmScATCSchedule
- myUplinkPeriod : FOSTimeInterval
+ GenerateLoad(FmScDAS): RWSlistCollectables
+ GenerateLtChgLoad(FmScDAS, RWSlistCollectables) RWSlistCollectables
is sent to FmLdLoadCatalog
creates/sends
FoLiATCLoad
CONTINUED
sends load info
is created by
receives
FmPcUplinkSchedReq
- myLoadName : RWString
- myNumOf Partitions : EcTInt
- mySizeOfLastPartition : EcTInt
- mySizeOfLoad : EcTInt
- myWindowInterval : FOSTimeInterval
FmMsLoadData
- myDirListAddr : EcTInt
- myDirectiveList : FmMnDirectiveList
- myLoadName : RWCString
- myUplinkWindow : FOSTimeInterval
is sent by
FmSmMapBuffer
+CreateConnection(EcTVoid): EcTInt
+DeleteBuffers(const RWSlistCollectables&) EcTVoid
+Destroy(EcTVoid) : EcTVoid
+GetATCBufStartTime(const FoEcTime&) FmMsATCBufferInfo
+MapATC(const FmMnDirectiveList&, const FOSTimeInterval&, const FoEcTime&, EcTInt) RWSlistCollectables&
+Receive(EcTVoid) : RWSlistCollectables&
+Send(const RWCollectable&): EcTVoid
+UpdateBuffer(const FmMsUpdateBuffer&) EcTVoid
{Proxy to FmSmSpacecraft}

Figure 3.2-7. Schedule Controller Object Model - Page 6

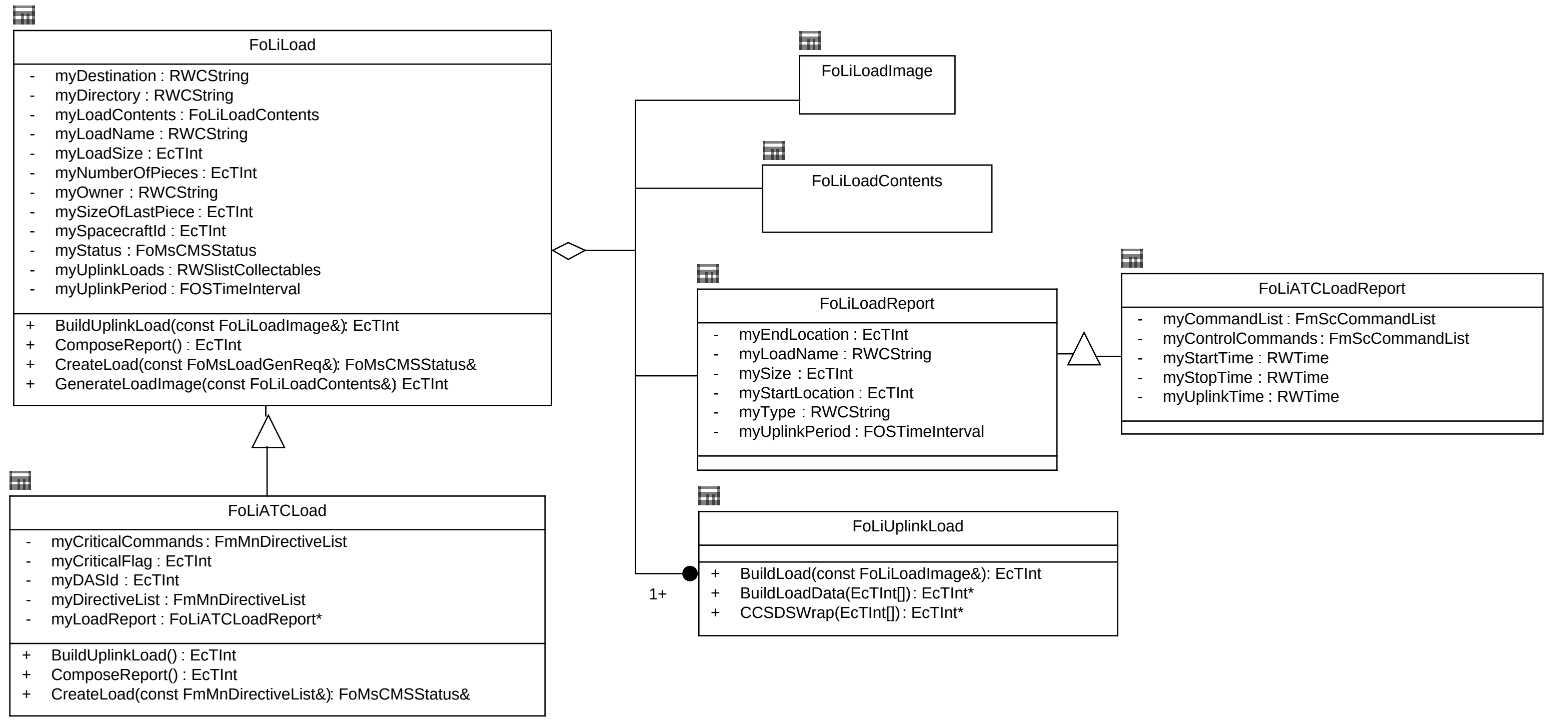
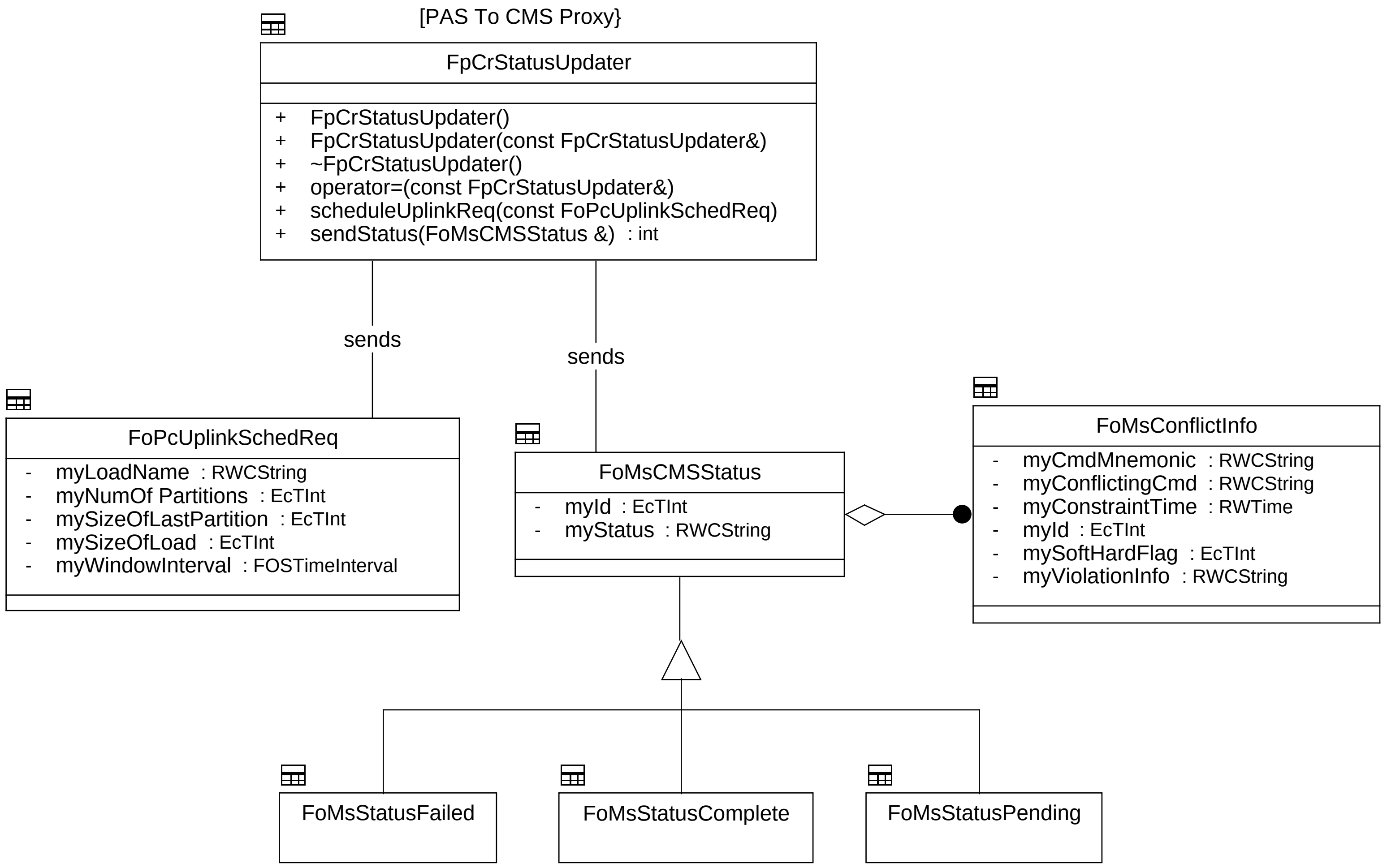

FoLiLoad
- myDestination : RWCString
- myDirectory : RWCString
- myLoadContents : FoLiLoadContents
- myLoadName : RWCString
- myLoadSize : EcTInt
- myNumberOfPieces : EcTInt
- myOwner : RWCString
- mySizeOfLastPiece : EcTInt
- mySpacecraftId : EcTInt
- myStatus : FoMsCMSStatus
- myUplinkLoads : RWSlistCollectables
- myUplinkPeriod : FOSTimeInterval
+ BuildUplinkLoad(const FoLiLoadImage&): EcTInt
+ ComposeReport() : EcTInt
+ CreateLoad(const FoMsLoadGenReq&): FoMsCMSStatus&
+ GenerateLoadImage(const FoLiLoadContents&) EcTInt
FoLiATCLoad
- myCriticalCommands : FmMnDirectiveList
- myCriticalFlag : EcTInt
- myDASId : EcTInt
- myDirectiveList : FmMnDirectiveList
- myLoadReport : FoLiATCLoadReport*
+ BuildUplinkLoad() : EcTInt
+ ComposeReport() : EcTInt
+ CreateLoad(const FmMnDirectiveList&): FoMsCMSStatus&
FoLiLoadImage
FoLiLoadContents
FoLiLoadReport
- myEndLocation : EcTInt
- myLoadName : RWCString
- mySize : EcTInt
- myStartLocation : EcTInt
- myType : RWCString
- myUplinkPeriod : FOSTimeInterval
FoLiATCLoadReport
- myCommandList : FmScCommandList
- myControlCommands : FmScCommandList
- myStartTime : RWTime
- myStopTime : RWTime
- myUplinkTime : RWTime
FoLiUplinkLoad
+ BuildLoad(const FoLiLoadImage&): EcTInt
+ BuildLoadData(EcTInt[]) : EcTInt*
+ CCSDSWrap(EcTInt[]) : EcTInt*
1+

Figure 3.2-8. Schedule Controller Object Model - Page 7



[PAS To CMS Proxy}
FpCrStatusUpdater
+ FpCrStatusUpdater()
+ FpCrStatusUpdater(const FpCrStatusUpdater&)
+ ~FpCrStatusUpdater()
+ operator=(const FpCrStatusUpdater&)
+ scheduleUplinkReq(const FoPcUplinkSchedReq)
+ sendStatus(FoMsCMSStatus &) : int
sends
sends
FoPcUplinkSchedReq
- myLoadName : RWCString
- myNumOf Partitions : EcTInt
- mySizeOfLastPartition : EcTInt
- mySizeOfLoad : EcTInt
- myWindowInterval : FOSTimeInterval
FoMsCMSStatus
- myId : EcTInt
- myStatus : RWCString
FoMsConflictInfo
- myCmdMnemonic : RWCString
- myConflictingCmd : RWCString
- myConstraintTime : RWTime
- myId : EcTInt
- mySoftHardFlag : EcTInt
- myViolationInfo : RWCString
FoMsStatusFailed
FoMsStatusComplete
FoMsStatusPending

Figure 3.2-9. Schedule Controller Object Model - Page 8

3.2.4 CMS Schedule Controller Dynamic Model

The Schedule Controller Dynamic Model described in this section consists of the following scenarios:

- Schedule Controller Initiation
- Detailed Activity Schedule Receipt with no constraint violations
- Detailed Activity Schedule Receipt with soft constraint violations
- Detailed Activity Schedule Receipt with hard constraint violations
- Late Change Receipt with soft constraint violations
- Constraint Check Only Receipt

3.2.4.1 Schedule Controller Initialization Scenario

3.2.4.1.1 Schedule Controller Initialization Scenario Abstract

The Schedule Controller Initialization scenario describes the process of initializing the Schedule Controller process.

3.2.4.1.2 Schedule Controller Initialization Summary Information

Interfaces:

- DMS
- Planning and Scheduling
- CMS Command Model
- CMS Spacecraft Model
- CMS Load Catalog
- CMS Ground Schedule

Stimulus:

- Schedule Controller is started

Desired Response:

- Establish connection to Planning and Scheduling
- Establish connection to Command Model
- Establish connection to Spacecraft Model
- Establish connection to Load Catalog
- Establish connection to Ground Schedule

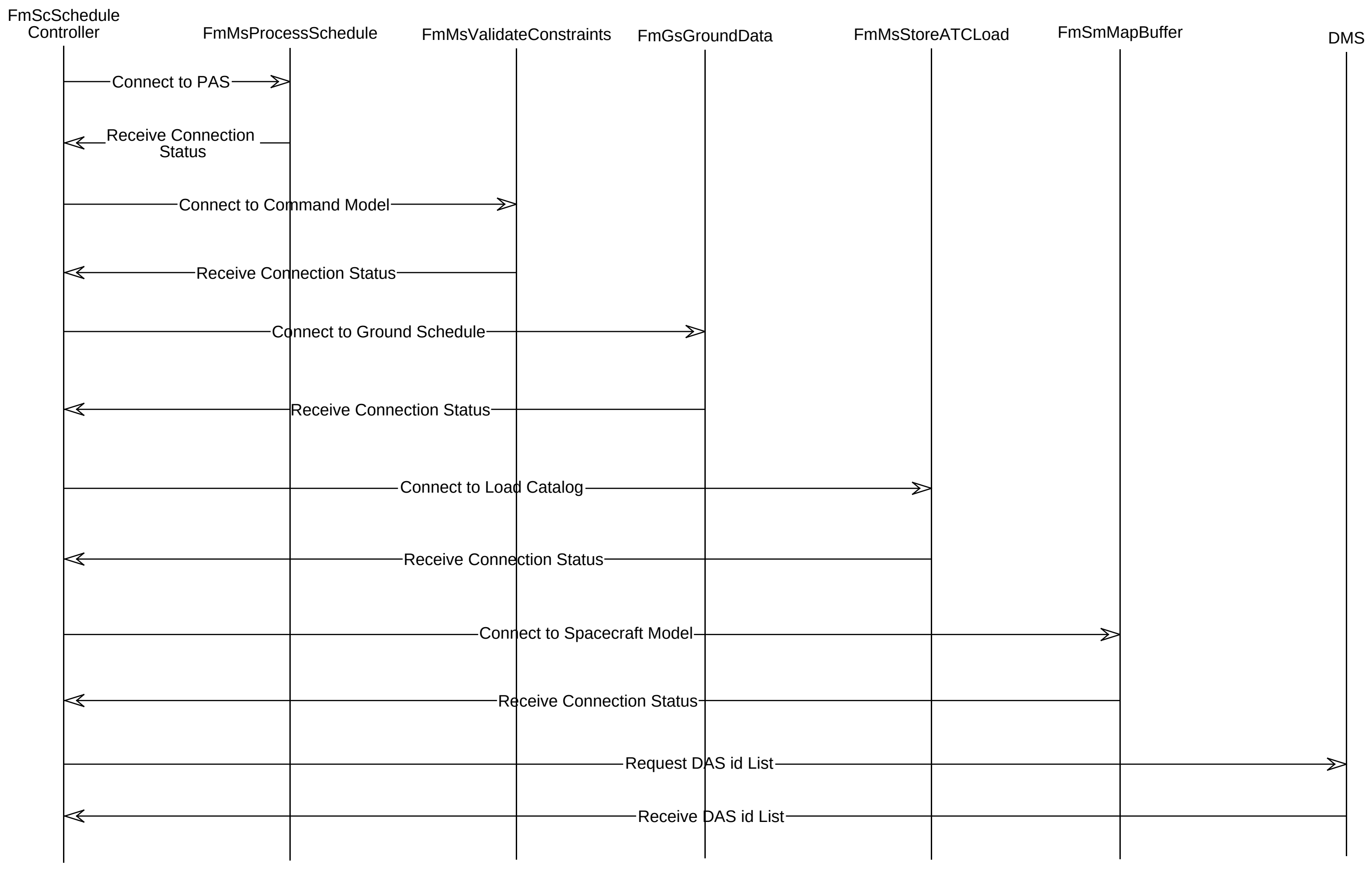


Figure 3.2-10. Schedule Controller Initialization Event Trace

- Retrieval of the DAS id list

Pre-Conditions:

- DMS software has been initiated
- Planning and Scheduling software has been initiated
- CMS Command Model software has been initiated
- CMS Spacecraft Model software has been initiated
- CMS Load Catalog software has been initiated
- CMS Ground Schedule software has been initiated

Post-Conditions:

- Schedule Controller up and running

3.2.4.1.3 Schedule Controller Initialization Scenario Description

The Schedule Controller Initialization Scenario event trace diagram is shown in Figure 3.2-10. The Schedule Controller will make the necessary connections to it's external entities. These are:

- Planning and Scheduling via the FmMsProcessSchedule proxy
- Load Catalog via the FmMsStoreATCLoad proxy
- Spacecraft Model via the FmSmMapBuffer proxy
- Ground Schedule via the FmGsGroundData proxy
- Command Model via the FmMsValidateContraints proxy

The Schedule Controller will also access the data base in order to retrieve the latest DAS id list. If this list is not available then the Schedule Controller will create an empty id list. After the connections are made and the DAS id list set-up, the Schedule Controller will be waiting to receive input from the PAS subsystem.

3.2.4.2 Detailed Activity Schedule Receipt with No Constraint Violations Scenario

3.2.4.2.1 Detailed Activity Schedule Receipt with No Constraint Violations Scenario Abstract

The Detailed Activity Schedule Receipt scenario describes the receipt of a Detailed Activity Schedule (DAS) from PAS, the expansion of the activities in the DAS into directives, requesting command-level constraint checking of the expanded list, generation of ATC loads, and merging the ground directives into the Ground Schedule.

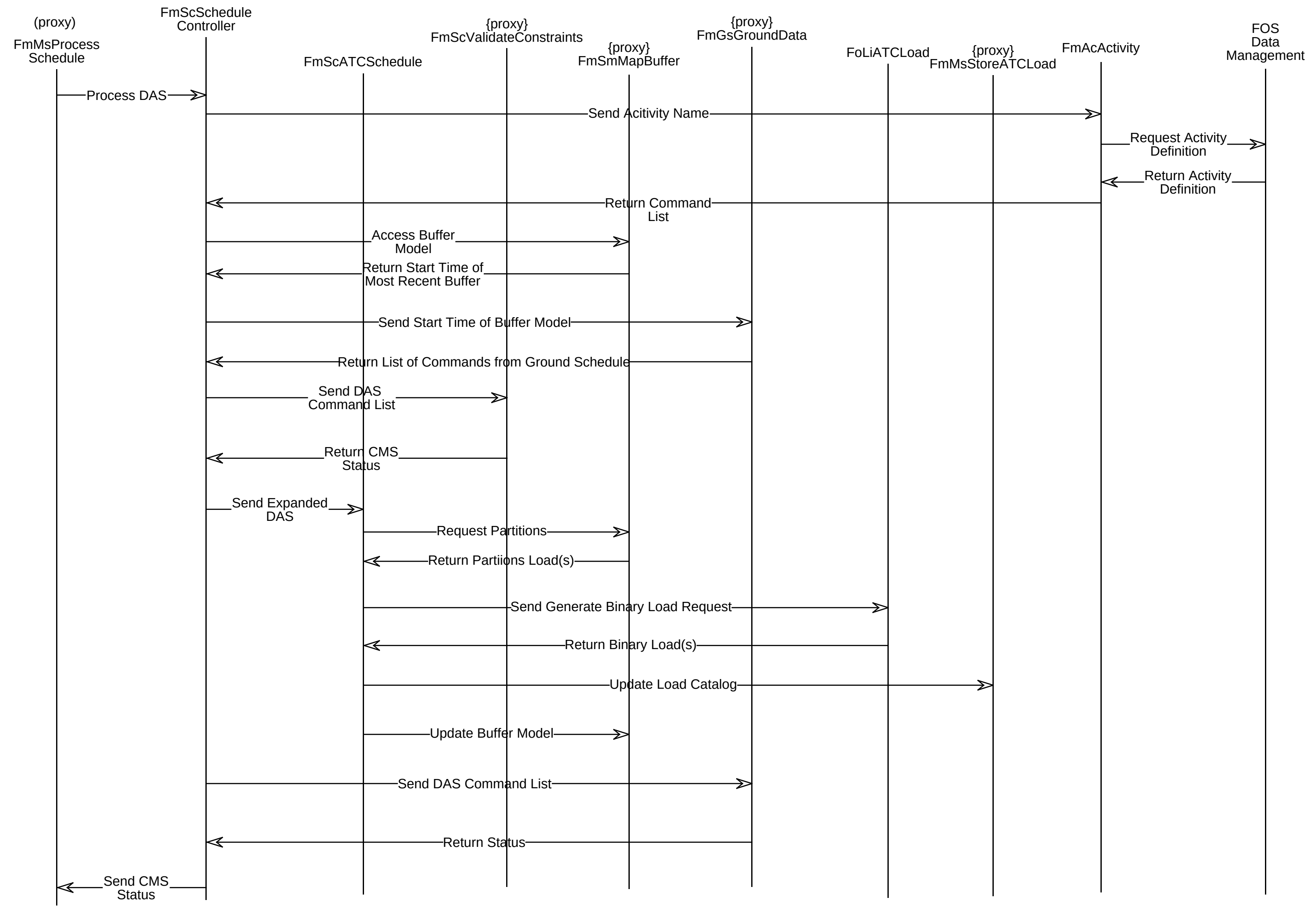


Figure 3.2-11. Schedule Controller DAS Receipt Event Trace - No Constraint Violations

3.2.4.2.2 Detailed Activity Schedule Receipt with No Constraint Violations Summary Information

Interfaces:

- Planning & Scheduling
- DMS
- CMS Command Model
- CMS Spacecraft Model
- CMS Load Catalog
- CMS Ground Schedule

Stimulus:

- Receipt of DAS Activity List

Desired Response:

- ATC load(s) for time span of DAS stored in ATC Load Catalog
- Ground Schedule updated with directives and orbital events for time span of DAS
- Status returned to PAS

Pre-Conditions:

- Schedule Controller software has been initiated

Post-Conditions:

- Updated list of processed Detailed Activity Schedules stored with DMS

3.2.4.2.3 Detailed Activity Schedule Receipt with No Constraint Violations Scenario Description

This scenario is shown in Figure 3.2-11 and can be broken down into 5 main functions:

- DAS receipt
- Activity Expansion
- Retrieval of Directives from the ATC Buffer Model for Continuity in Constraint Checking
- Constraint Checking
- ATC Load Generation

The Schedule Controller receives a Detailed Activity Schedule activity list object from PAS and instantiates a DAS Directive List object to receive the expanded activities.

For each activity in the Activity List, the Schedule Controller instantiates an Activity object and invokes its expand function. The Activity instantiates an Activity Definition object which retrieves the activity definition from DMS. Activity converts each command in the database definition of the activity to a Directive object of the appropriate type. Each Directive is merged in time order into the DAS Command List.

Once all activities in the DAS have been expanded into directives, the Schedule Controller retrieves space directives from the ATC Buffer Model. The Schedule Controller instantiates a constraint check list and the Spacecraft Model adds the directives from the most recent ATC buffer to this constraint check list. These directives are needed in order to maintain continuity in constraint checking the DAS. The constraint check list will then merge the directives from expanding the DAS into itself, and this constraint check list is what will be constraint checked.

Next the Schedule Controller sends a message to the Command Model object to constraint check the merged constraint check list. The Command Model object will iterate through each command in the command list. Each command can have an optional constraint rule associated with it. Each rule is identified and the associated command is constraint checked based on the rule. Since no constraints were found when constraint checking the DAS, processing continues without interruption.

In order to build the load, Schedule Controller invokes the create load function of ATC Schedule. The create load function accepts a directive list that contains a list of the space directives from the expansion of the DAS activities. ATC Schedule invokes the Spacecraft Model via a proxy. The Spacecraft model computes the number of partitions that the load will need to be broken into based on the uplink period and the available space in the ATC buffer. The Spacecraft Model creates a working ATC buffer model for each of the partitions and returns a list of Load Data objects. Each Load Data object contains the partitioned load and an associated uplink window. For each Load Data object in the list, ATC Schedule creates an ATC Load object and invokes the create load function in ATC Load. ATC Load builds the binary load and the load report from the input list of directives. ATC Schedule then updates the load catalog via the Store ATC Load proxy. Store ATC Load creates an entry for the generated load in the load catalog. After the load catalog is updated, ATC Schedule promotes the working ATC buffer model to a predicted ATC buffer models via the Spacecraft Model proxy.

Finally the Schedule Controller invokes the DAS processing function of Ground Schedule. The Ground Schedule merges the Directives in the DAS Command List into its time ordered directive list.

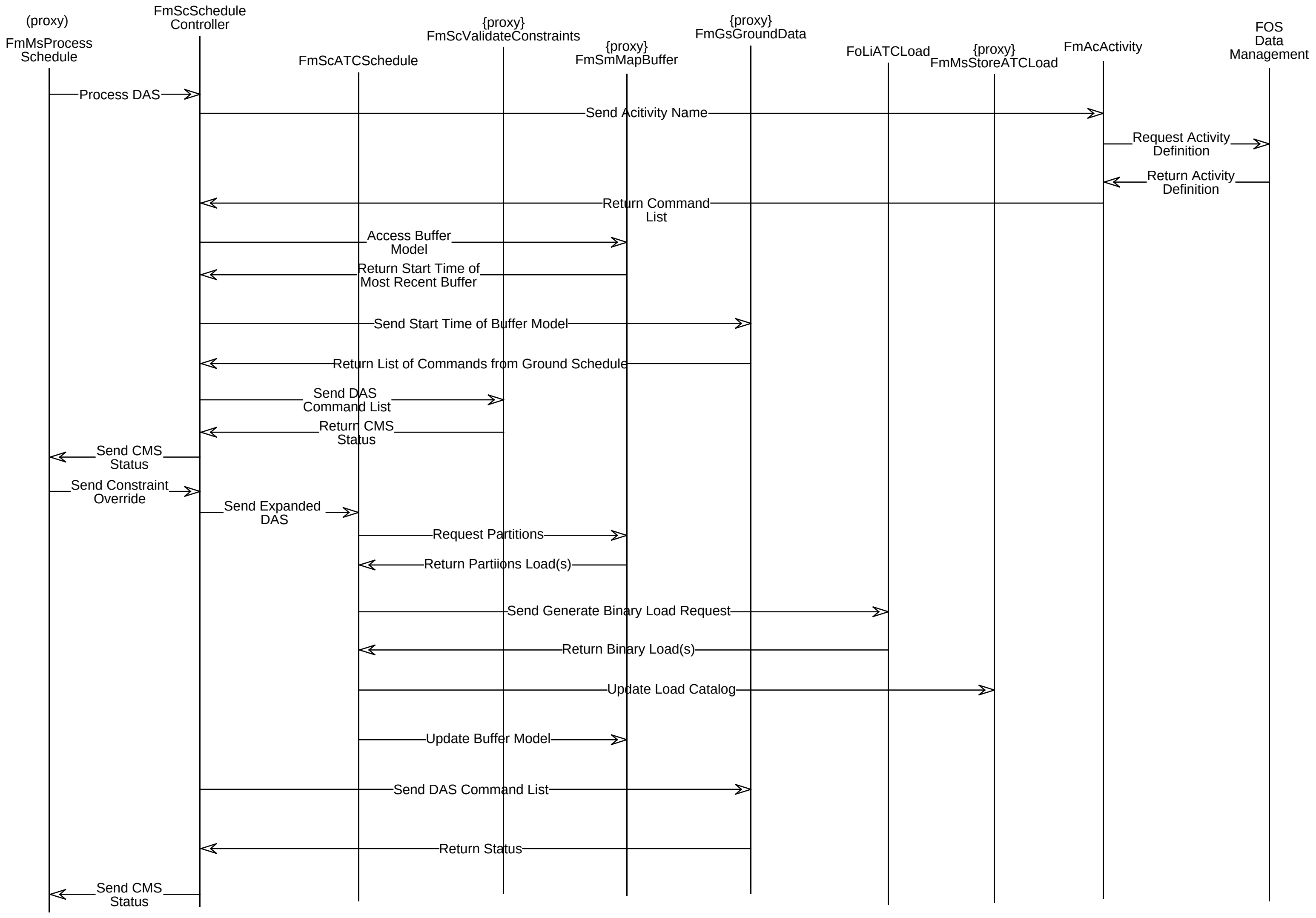


Figure 3.2-12. DAS Receipt Scenario - Soft Constraint Violation

3.2.4.3 Detailed Activity Schedule Receipt with Soft Constraint Violations Scenario

3.2.4.3.1 Detailed Activity Schedule Receipt with Soft Constraint Violations Scenario Abstract

The Detailed Activity Schedule Receipt scenario describes the receipt of a Detailed Activity Schedule (DAS) from PAS, the expansion of the activities in the DAS into directives, requesting command-level constraint checking of the expanded list, sending a soft constraint status to PAS, receiving an override soft constraint message from PAS, generation of ATC loads, and merging the ground directives into the Ground Schedule.

3.2.4.3.2 Detailed Activity Schedule Receipt with Soft Constraint Violations Summary Information

Interfaces:

- Planning & Scheduling
- DMS
- CMS Command Model
- CMS Spacecraft Model
- CMS Load Catalog
- CMS Ground Schedule

Stimulus:

- Receipt of DAS Activity List

Desired Response:

- Find soft constraint, send soft constraint status to PAS and receive override message from PAS
- ATC load(s) for time span of DAS stored in ATC Load Catalog
- Ground Schedule updated with directives and orbital events for time span of DAS
- Status returned to PAS

Pre-Conditions:

- Schedule Controller software has been initiated

Post-Conditions:

- Updated list of processed Detailed Activity Schedules stored with DMS

3.2.4.3.3 Detailed Activity Schedule Receipt with Soft Constraint Violation Scenario Description

This scenario is shown in Figure 3.2-12 and can be broken down into 5 main functions:

- DAS receipt
- Activity Expansion
- Retrieval of Directives from the ATC Buffer Model for Continuity in Constraint Checking
- Constraint Checking
- ATC Load Generation

The Schedule Controller receives a Detailed Activity Schedule activity list object from PAS and instantiates a DAS Directive List object to receive the expanded activities.

For each activity in the Activity List, the Schedule Controller instantiates an Activity object and invokes its expand function. The Activity instantiates an Activity Definition object which retrieves the activity definition from DMS. Activity converts each command in the database definition of the activity to a Directive object of the appropriate type. Each Directive is merged in time order into the DAS Command List.

Once all activities in the DAS have been expanded into directives, the Schedule Controller retrieves space directives from the ATC Buffer Model. The Schedule Controller instantiates a constraint check list and the Spacecraft Model adds the directives from the most recent ATC buffer to this constraint check list. These directives are needed in order to maintain continuity in constraint checking the DAS. The constraint check list will then merge the directives from expanding the DAS into itself, and this constraint check list is what will be constraint checked.

Next the Schedule Controller sends a message to the Command Model object to constraint check the merged constraint check list. The Command Model object will iterate through each command in the command list. Each command can have an optional constraint rule associated with it. Each rule is identified and the associated command is constraint checked based on the rule. For a soft constraint is violation, a CMS Pending Status object is sent to PAS and DAS processing waits for a response from PAS on whether to continue to process the DAS with soft constraints or to cease processing. Upon the Schedule Controller receiving a override soft constraint indication, processing continues.

In order to build the load, Schedule Controller invokes the create load function of ATC Schedule. The create load function accepts a directive list that contains a list of the space directives from the expansion of the DAS activities. ATC Schedule invokes the Spacecraft Model via a proxy. The Spacecraft model computes the number of partitions that the load will need to be broken into based on the uplink period and the available space in the ATC buffer. The Spacecraft Model creates a working ATC buffer model for each of the partitions and returns a list of Load Data objects. Each Load Data object contains the partitioned load and an associated uplink window. For each Load Data object in the list, ATC Schedule creates an ATC Load object and invokes the create load function in ATC Load. ATC Load builds the binary load and the load report from the input list of directives. ATC Schedule then updates the load catalog via the Store ATC Load proxy. Store ATC Load creates an entry for the generated load in the load catalog. After the load catalog is updated, ATC Schedule promotes the working ATC buffer model to a predicted ATC buffer models via the Spacecraft Model proxy.

Finally the Schedule Controller invokes the DAS processing function of Ground Schedule. The Ground Schedule merges the Directives in the DAS Command List into its time ordered directive list.

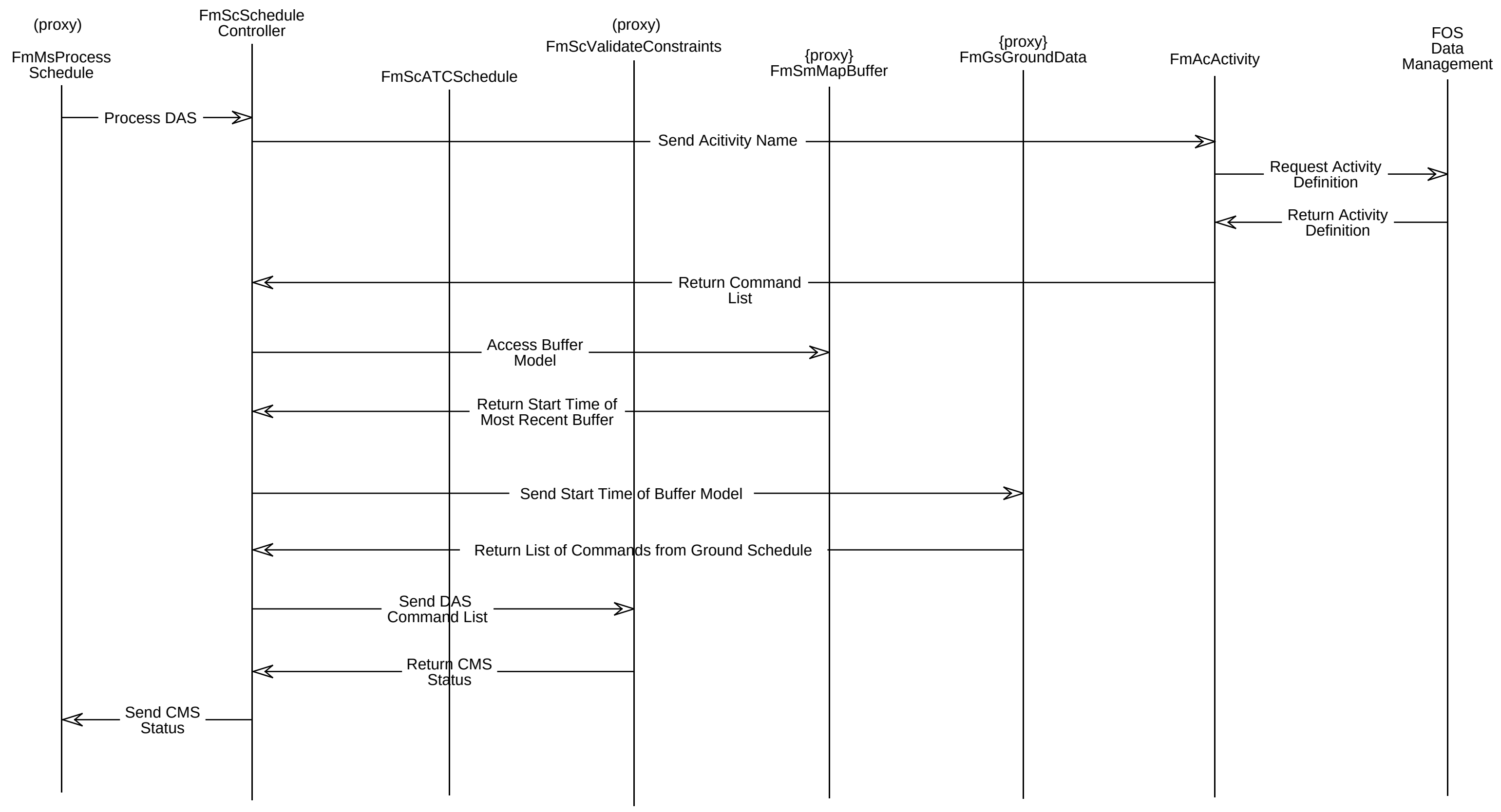


Figure 3.2-13. DAS Receipt Scenario - Hard Constraint Violation

3.2.4.4 Detailed Activity Schedule Receipt with Hard Constraint Violations Scenario

3.2.4.4.1 Detailed Activity Schedule Receipt with Hard Constraint Violations Scenario Abstract

The Detailed Activity Schedule Receipt scenario describes the receipt of a Detailed Activity Schedule (DAS) from PAS, the expansion of the activities in the DAS into directives, requesting command-level constraint checking of the expanded list, sending a hard constraint status to PAS.

3.2.4.4.2 Detailed Activity Schedule Receipt with Hard Constraint Violations Summary Information

Interfaces:

- Planning & Scheduling
- DMS
- CMS Command Model
- CMS Spacecraft Model

Stimulus:

- Receipt of DAS Activity List

Desired Response:

- Find hard constraint, and send hard constraint status to PAS.

Pre-Conditions:

- Schedule Controller software has been initiated

Post-Conditions:

- none

3.2.4.4.3 Detailed Activity Schedule Receipt with Hard Constraint Violation Scenario Description

This scenario is shown in Figure 3.2-13 and can be broken down into 4 main functions:

- DAS receipt
- Activity Expansion
- Retrieval of Directives from the ATC Buffer Model for Continuity in Constraint Checking
- Constraint Checking

The Schedule Controller receives a Detailed Activity Schedule activity list object from PAS and instantiates a DAS Directive List object to receive the expanded activities.

For each activity in the Activity List, the Schedule Controller instantiates an Activity object and invokes its expand function. The Activity instantiates an Activity Definition object which retrieves the activity definition from DMS. Activity converts each command in the database definition of the activity to a Directive object of the appropriate type. Each Directive is merged in time order into the DAS Command List.

Once all activities in the DAS have been expanded into directives, the Schedule Controller retrieves space directives from the ATC Buffer Model. The Schedule Controller instantiates a constraint check list and the Spacecraft Model adds the directives from the most recent ATC buffer to this constraint check list. These directives are needed in order to maintain continuity in constraint checking the DAS. The constraint check list will then merge the directives from expanding the DAS into itself, and this constraint check list is what will be constraint checked.

Next the Schedule Controller sends a message to the Command Model object to constraint check the merged constraint check list. The Command Model object will iterate through each command in the command list. Each command can have an optional constraint rule associated with it. Each rule is identified and the associated command is constraint checked based on the rule. When a hard constraint is violated, a CMS Failed Status object is sent to PAS and processing of the DAS ceases.

3.2.4.5 Late Change Receipt Scenario

3.2.4.5.1 Late Change Receipt Scenario Abstract

The Late Change Receipt scenario describes the receipt of a Detailed Activity Schedule containing late change information from PAS, the deletion of loads and ground directives generated from the same DAS when it was previously processed and any subsequent DASs already processed, the expansion of the activities in the DAS into directives, command-level constraint checking of the expanded list, generation of ATC loads, and merging the ground directives into the Ground Schedule.

3.2.4.5.2 Late Change Receipt Summary Information

Interfaces:

- Planning & Scheduling
- DMS
- CMS Command Model
- CMS Spacecraft Model
- CMS Load Catalog
- CMS Ground Schedule

Stimulus:

- Receipt of Late Change Activity List

Desired Response:

- ATC load(s) built from late change DAS and subsequent DASs deleted from DMS

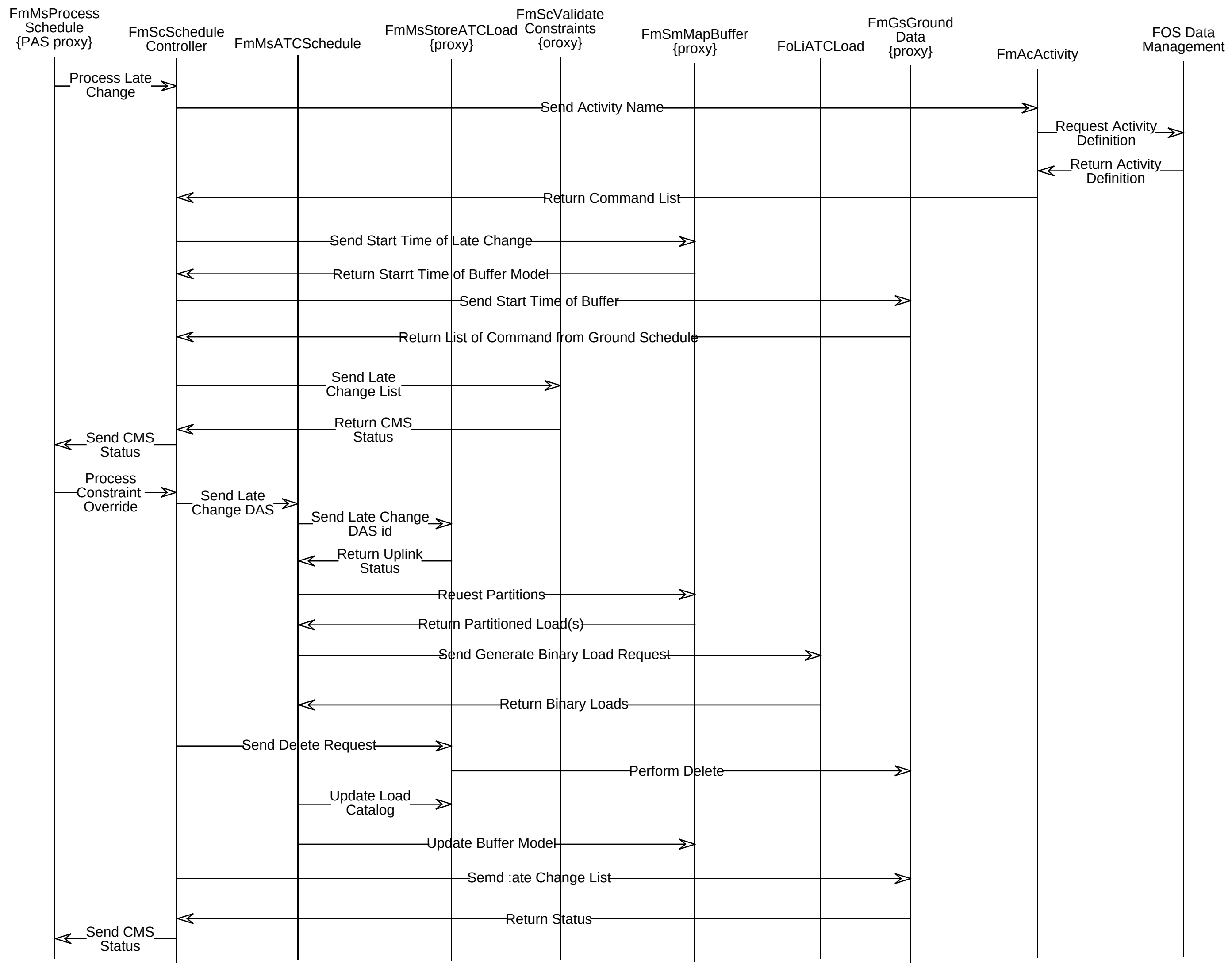


Figure 3.2-14. Late Change Receipt Event Trace

- Directives from late change DAS and subsequent DASs deleted from Ground Schedule
- ATC load(s) for time span of DAS stored in ATC Load Catalog
- Ground Schedule updated with directives and orbital events for time span of DAS
- Status returned to PAS

Pre-Conditions:

- Schedule Controller software has been initiated
- Spacecraft Model software has been initiated
- Load Catalog software has been initiated
- Ground Schedule software has been initiated
- Command Model software has been initiated

Post-Conditions:

- Updated list of processed Detailed Activity Schedules stored with DMS

3.2.4.5.3 Late Change Receipt Scenario Description

This scenario is shown in Figure 3.2-14 and can be broken down into 6 main functions:

- Late Change receipt
- Activity Expansion
- Retrieval of Directives from the ATC Buffer Model for Continuity in Constraint Checking
- Constraint Checking
- ATC Load Generation
- Deletion of CMS Schedule Products

The Schedule Controller receives a Late Change Detailed Activity Schedule activity list object from PAS and instantiates a DAS Directive List object to receive the expanded activities.

For each activity in the Activity List, the Schedule Controller instantiates an Activity object and invokes its expand function. The Activity instantiates an Activity Definition object which retrieves the activity definition from DMS. Activity converts each command in the database definition of the activity to a Directive object of the appropriate type. Each Directive is merged in time order into the DAS Command List.

Once all activities in the DAS have been expanded into directives, the Schedule Controller retrieves space directives from the ATC Buffer Model. The Schedule Controller instantiates a constraint check list and the Spacecraft Model adds the directives from the most recent ATC buffer to this constraint check list. These directives are needed in order to maintain continuity in constraint checking the DAS. The constraint check list will then merge the directives from expanding the DAS into itself, and this constraint check list is what will be constraint checked.

Next the Schedule Controller sends a message to the Command Model object to constraint check the merged constraint check list. The Command Model object will iterate through each command in the command list. Each command can have an optional constraint rule associated with it. Each rule is identified and the associated command is constraint checked based on the rule. Since no constraints were found when constraint checking the DAS, processing continues without interruption.

In order to build the load, Schedule Controller invokes the create load function of ATC Schedule. The create load function accepts a directive list that contains a list of the space directives from the expansion of the DAS activities. ATC Schedule invokes the Spacecraft Model via a proxy. The Spacecraft model computes the number of partitions that the load will need to be broken into based on the uplink period and the available space in the ATC buffer. The Spacecraft Model creates a working ATC buffer model for each of the partitions and returns a list of Load Data objects. Each Load Data object contains the partitioned load and an associated uplink window. For each Load Data object in the list, ATC Schedule creates an ATC Load object and invokes the create load function in ATC Load. ATC Load builds the binary load and the load report from the input list of directives. ATC Schedule then updates the load catalog via the Store ATC Load proxy. Store ATC Load creates an entry for the generated load in the load catalog. After the load catalog is updated, ATC Schedule promotes the working ATC buffer model to a predicted ATC buffer models via the Spacecraft Model proxy.

The Schedule Controller creates a list of DAS ids that have been previously processed and have starting times after the late change DAS. These DAS ids are then removed from the list of processed DASs maintained by the Schedule Controller. The Schedule Controller sends the list of DAS id that are after the Late Change to Load Catalog. Load Catalog deletes the ATC loads that were generated from those DASs and their associated entries from the Load Catalog. Next the Spacecraft Model will delete the ATC buffer models associated with each DAS id in the list the Ground Schedule deletes all Directives associated with those DASs from the ground schedule.

Finally the Schedule Controller invokes the DAS processing function of Ground Schedule. The Ground Schedule merges the Directives in the DAS Command List into its time ordered directive list.

3.2.4.6 "What-if" Scenario

3.2.4.6.1"What-if" Scenario Abstract

The Constraint Check Receipt scenario describes the receipt of a Detailed Activity Schedule containing activities to be constraint checked, the expansion of the activities in the DAS into directives, and command-level constraint checking of the expanded list.

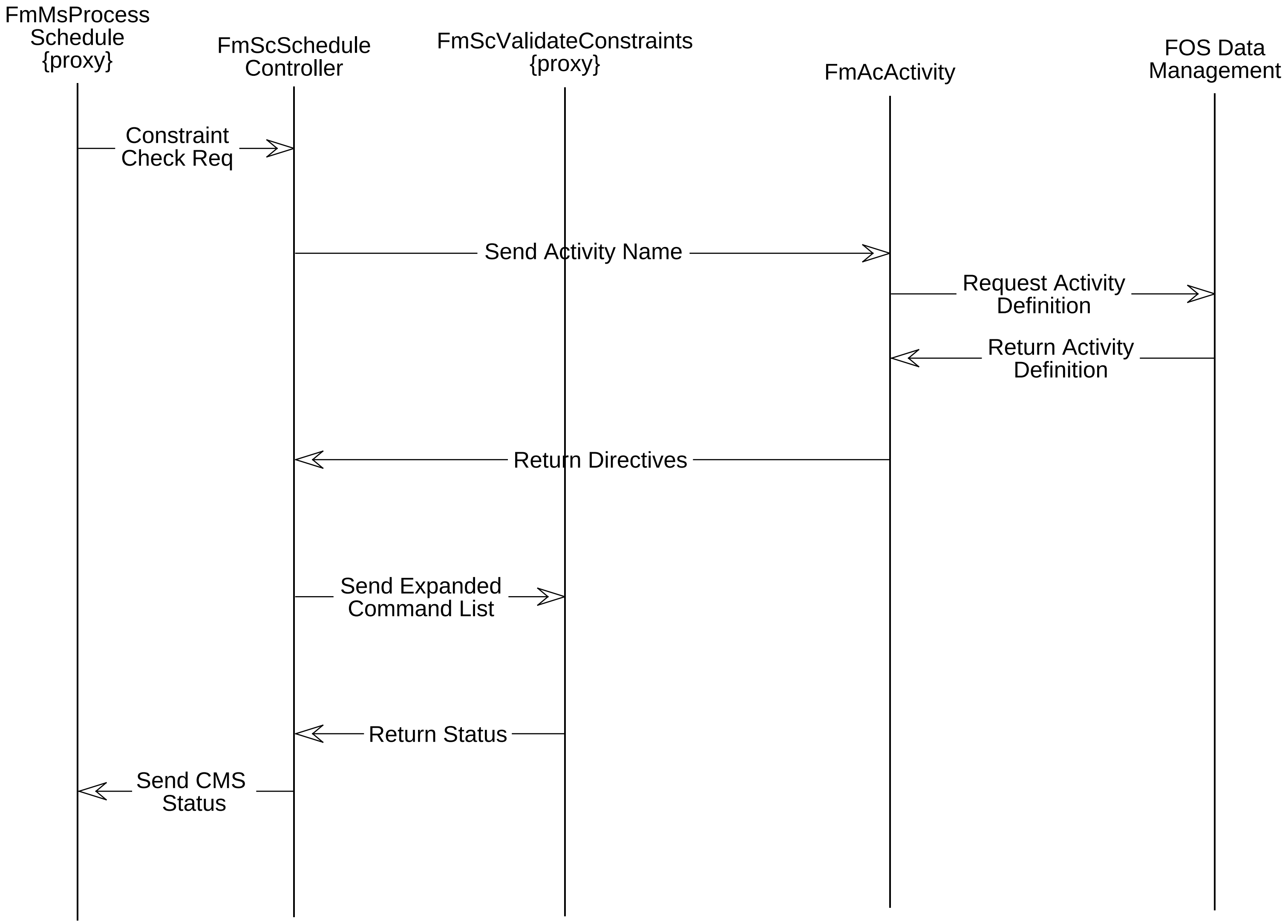

FmMsProcess Schedule {proxy}
FmScSchedule Controller
FmScValidateConstraints {proxy}
FmAcActivity
FOS Data Management
Constraint Check Req
Send Activity Name
Request Activity Definition
Return Activity Definition
Return Directives
Send Expanded Command List
Return Status
Send CMS Status

Figure 3.2-15. "What-if" Receipt Scenario - Hard Constraint Violation

3.2.4.6.2"What-if" Receipt Summary Information

Interfaces:

Planning & Scheduling

DMS

CMS Command Model

Stimulus:

Receipt of Constraint Check Activity List

Desired Response:

Status returned to PAS

Pre-Conditions:

Schedule control software has been initiated

Command Model software has been initiated

Post-Conditions:

none

3.2.4.6.3"What-if" Receipt Scenario Description

The event trace diagram for this scenario is in Figure 3.2-15 and can be broken down into 3 main functions:

Constraint Check receipt

Activity Expansion

Constraint Checking

The Schedule Controller receives a "What-if" activity list object from PAS and instantiates a DAS constraint check list object to receive the expanded activities.

For each activity in the "What-if" Activity List, the Schedule Controller instantiates an Activity object and invokes its expand function. The Activity instantiates an Activity Definition object which retrieves the activity definition from DMS. Activity converts each command in the database definition of the activity to a Directive object of the appropriate type. Each Directive is merged in time order into the DAS Command List.

Once the expansion of a activity schedule is complete, the Schedule Controller sends a message to the Command Model object to constraint check the expanded command list. The Command Model iterates through each command in the command list. Each command can have an optional constraint rule associated with it. Each rule is identified and the associated command is constraint checked based on the rule. A CMS Status object is sent back to PAS which indicates if any constraints were violated. Load Generation is NOT invoked for a constraint checking only scenario.

3.2.5 CMS Schedule Controller Data Dictionary

FmScScheduleController

`class` **`FmScScheduleController`**

The main controller in the building of an ATC load from an activity list

Public Functions

`EcTVoid` **`MessageHandler`**`()`

Receives an activity list as input, decides what type of object it is, and distributes it to the proper operation

Private Functions

`EcTVoid` **`InitializeController`**`()`

Initializes the Schedule Controller

`EcTVoid` **`ProcessActivities`**`(FoScActivityList)`

Processes an incoming activity list from PAS

`EcTVoid` **`ProcessDeleteReq`**`(EcTInt)`

Processes an incoming delete load request from PAS

`EcTVoid` **`SendEventMessage`**`(FoEvEvent)`

Sends an event to the event handler

`EcTVoid` **`SendStatus`**`(const FoMsCMSStatus)`

Sends a ground script generation status to the FUI proxy

`EcTVoid` **`SendUplinkSchedReq`**`(FmPcUplinkSchedReq)`

Sends the a list of uplink schedule requests to PAS via the proxy

Private Data

`RWSlistCollectables` **`myDASidList`**

List of DAS ids

`FoEvEvent*` **`myEventPtr`**

Pointer to an event to be sent to the event handler

FmMsProcessSchedule

`class` **`FmMsProcessSchedule`**

Proxy for PAS to CMS Schedule Controller

Public Functions

`FoMsCMSStatus` **`ConstraintOverride`**`(enum(y, n))`

Sends the constraint override flag to the schedule controller

`EcTInt` **`DeleteLoads`**`(EcTInt)`

Deletes the load(s) for the associated input DAS id

`EcTVoid` **`ProcessActSchedule`**`(FoScActivityList)`

Calls operation to send the activity schedule to the schedule controller

Private Functions

`EcTInt` **`Connect`**`()`

Makes the IPC connection to the schedule controller

`EcTVoid` **`Disconnect`**`()`

Disconnects the IPC connection from the schedule controller

`EcTVoid` **`SendActSched`**`(FoScActivityList)`

Sends the activity schedule to the schedule controller via IPC

`EcTInt` **`SendDeleteReq`**`(EcTInt)`

Sends a delete load request for a specified DAS to the schedule controller via IPC

FmScATCSchedule

`class` **`FmScATCSchedule`**

Responsible for coordinating the building of the ATC load

Base Classes

`public` **`FmScSchedule`**

Public Functions

`RWSlistCollectables` **`GenerateLoad`**`(FmMnDirectiveList)`

This operation that will coordinate the following for DAS processing:

```
- Partitioning of the load if necessary
- Creation of the load and load report
- Updating the load catalog
- Updating the buffer model
- Creating an uplink request
```

`RWSlistCollectables` **`GenerateLtChgLoad`**`(const FmMnDirectiveList&, const RWSlistCollectables)`

This operation that will coordinate the following for Late change processing:

```
- Partitioning of the load if necessary
- Creation of the load and load report
- Updating the load catalog
- Updating the buffer model
- Creating an uplink request
- Removing entries from the load catalogs and their associated loads
```

Private Data

`FOSTimeInterval` **`myUplinkPeriod`**

Requested uplink period for the load

FmPcUplinkSchedReq

`class` **`FmPcUplinkSchedReq`**

Uplink request to be sent to PAS

Private Data

`RWCString` **`myLoadName`**

Is the name of the load generated

`EcTInt` **`myNumOfPartitions`**

Indicates the number of 4k partitions the load was broken into

`EcTInt` **`mySizeOfLastPartition`**

Indicates the size of the last partition

`EcTInt` **`mySizeOfLoad`**

Indicates the size of the entire load

`FOSTimeInterval` **`myWindowInterval`**

Indicates the recommended uplink window

FmAcActivity

`class` **`FmAcActivity`**

Responsible for expanding an activity into a group of command based on the stored activity definition

Public Functions

`RWDlistCollectables` **`ExpandActivity`**`(FoScActivity&)`

This operation does the expansion of each activity

FmExProcedure

`class` **`FmExProcedure`**

responsible for the expansion of procedures

Public Functions

`EcTInt` **`ExpandProcedure`**`(RWDlistcollectables)`

Performs the expansion of a procedure and adds the commands to the DAS expanded command list

`RWtime` **`GetExecTime`**`()`

Returns the value of MyExecTime

Private Data

`RWTime` **`MyExecTime`**

Time of the execute procedure command

`RWString` **`MyFilename`**

Filename of the procedure to be executed

FmExRTS

`class` **`FmExRTS`**

responsible for the expansion of an RTS

Public Functions

`EcTInt` **`ExpandRTS`**`(RWDlistCollectables, RWDlistCollectables)`

This operation expands the RTS into a set of space directives

FmMnDirectiveList

`class` **`FmMnDirectiveList`**

List of FoEcDirectives

Base Classes

`public` **`RWDlistCollectables`**

Public Functions

`EcTVoid` **`CreateExpandedList`**`(FoScActivityList)`

Expands the list of activities into a list of directives

`FoMsCMSStatus` **`FindConstraints`**`(void)`

Calls the necessary operations to perform constraint checking on the expanded directive list

`EcTVoid` **`MergeWithList`**`(const RWDlistCollectables)`

Merges an input list into the directive list

Private Data

`EcTInt` **`myId`**

DASid that this command list was generated from

`RWTime` **`myStartTime`**

Start time of expanded command list

`RWTime` **`myStopTime`**

Stop time of expanded command list

FmScDAS

`class` **`FmScDAS`**

Base Classes

`public` **`FmMnDirectiveList`**

Public Types

`class` **`FmScDAS`**

A description of the class

Base Classes

`public` **`FmMnDirectiveList`**

Public Functions

`FoMsCMSStatus` **`FindConstraints`**`(void)`

Gets the buffer start time and the commands from the ground schedule then calls the base class operation to perform the constraint checks

`RWSlistCollectables` **`GetPartitions`**`(const RWTime)`

Calls the spacecraft model proxy to get the number of partitions the directive list must be broken into in order to fit into the spacecraft buffer

FmScConstCk

`class FmScConstCk`

Directives to be constraint checked

Base Classes

`public FmScCommandList`

Public Functions

`EcTVoid MergeWithFilter(const RWDlistCollectables)`

Takes as input the expanded sorted command list and builds a filtered command list to be constraint checked

FoMsCMSStatus

`class FoMsCMSStatus`

status for processing

Private Data

`EcTInt myId`

The id of this message.

`RWCString myStatus`

Pertinent information about the status object. Mostly used to explain why a process failed.

FoMsConflictInfo

`class FoMsConflictInfo`

Base Classes

`public RWCollectable`

Private Data

`RWCString myCmdMnemonic`

The mnemonic of this command.

`RWCString myConflictingCmd`

The mnemonic of the command with which this command conflicts.

`RWTime myConstraintTime`

The time at which the conflicting command is scheduled.

`EcTInt myId`

For a DAS, this will be an activity id. For RTS and procedures, this will be the line number of this command in the contents file.

`EcTInt mySoftHardFlag`

An indicator specifying a hard or soft constraint.

`RWCString myViolationInfo`

Textual information concerning the violation.

FmGsGroundData

class **FmGsGroundData**

Base Classes

`public` **`RWCollectable`**

Public Functions

`EcTVoid` **`DeleteDirectives`**`(const RWSlistCollectables&)`

Called by the Schedule Controller to delete certain directives from the Ground Schedule.

`EcTVoid` **`DeliverDirectives`**`(const FmMnDirectiveList&)`

Called by the Schedule Controller to put the directives from a DAS into the Ground Schedule.

`FmMnDirectiveList` **`ReturnCCList`**`(const RWTime&, const RWSListCollectables&)`

Called by the Schedule Controller to request a list of certain directives from the Ground Schedule.

Private Functions

`EcTInt` **`CreateConnection`**`()`

Creates a connection between this proxy and the Ground Schedule.

`EcTVoid` **`DestroyConnection`**`()`

Destroys the connection between this proxy and the Ground Schedule.

`RWCollectable` **`Receive`**`()`

Receives an object by IPC from the Ground Schedule.

`EcTVoid` **`Send`**`(const RWCollectable&)`

IPC's an object to the Ground Schedule.

FmMsStoreATCLoad

class **FmMsStoreATCLoad**

class definition - This class represents an interface between the ATC Schedule and the Load Catalog. It uses IPC to relay information between this class and the Load Catalog. ATC Schedule sends information to this class via function calls.

Public Functions

`EcTInt` **`CheckForLoad`**`(EcTInt)`

Called by ATC Schedule to send a DAS Id to the Load Catalog and get back an integer status, indicating that the load associated with this DAS Id has or has not been uplinked.

`EcTInt` **`CreateConnection`**`()`

Creates the two-way connection between this proxy and FmLdLoadCatalog.

`EcTInt` **`DeleteLoads`**`(const RWSlistCollectables&)`

Called by ATC Schedule to delete all loads on the input list. Returns a response.

`EcTVoid` **`DestroyConnection`**`()`

Destroys the connection between this proxy and FmLdLoadCatalog.

`EcTInt` **`Receive`**`()`

Receives an FoMsCMSStatus object from FmLdLoadCatalog via IPC and returns it.

`EcTVoid` **`Send`**`(const RWCollectable&)`

Sends an object to FmLdLoadCatalog via IPC.

`EcTInt` **`StoreLoad`**`(const FoLiATCLoad&)`

Called by ATC Schedule to send a load for storage. Returns a response.

FmMsValidateConstraints

`class` **`FmMsValidateConstraints`**

This class represents the interface proxy class between CMS internal subsystems and the FmCcCommandModel class. FmC-cCommandModel manages the command rule-based constraint checking.

Public Functions

`EcTInt` **`CreateConnection`**`(void)`

Establishes a connection with FmCcCommandModel to receive constraint checking request from the schedule controller and the load catalog

`EcTVoid` **`DestroyConnection`**`(void)`

Destroys the connection with FmCcCommandModel

`FoMsCMSStatus&` **`Receive`**`(void)`

Receives the results of rule-base command constraint checking, FoMsCMSStatus

`FoMsCMSStatus&` **`Send`**`(const RWCollectable&)`

Sends either a FmScConstCk command list from the schedule controller or a FoEcDirective list created from an RTS load contents file to the FmCcCommandModel for rule-base command constraint checking

`FoMsCMSStatus&` **`ValidateCommands`**`(const FmScConstCk&)`

FmScScheduleController invokes this function to send the DAS scheduled command list to be command rule-based constraint checked

`FoMsCMSStatus&` **`ValidateRTS`**`(const RWCString&, const RWCString&)`

FmLdLoadCatalog invokes this function to send the directory name and load name from the generate RTS load request to be command rule-based constraint checked. This function creates the FoCcDirectiveList to the FmCcCommandModel.

FmSmMapBuffer

`class` **`FmSmMapBuffer`**

This class represents the interface proxy class between CMS internal subsystems and the FmSmSpacecraft class. FmSmSpace-craft manages the buffer modeling for ATC, RTS and table buffers and the ground

`imaging.`

Public Functions

`EcTInt` **`CreateConnection`**`(EcTVoid)`

Establishes a connection with FmSmSpacecraft to receive requests from the schedule controller and the load catalog

`EcTVoid` **`DeleteBuffers`**`(const RWSlistCollectables&)`

Request received from load catalog when a late change as been successfully processed. The predicted buffer models as-sociated with all of the generated loads are deleted. Instantiates an FmMsDeleteATCBuffers object.

`EcTVoid` **`Destroy`**`(EcTVoid)`

Destroys the connection with FmCcCommandModel

`FmMsATCBufferInfo` **`GetATCBufStartTime`**`(const FoEcTime&)`

Requests the start time of the 1st command in the buffer that will be used to model the newly received DAS or late change request

`RWSlistCollectables&` **`MapATC`**`(const FmMnDirectiveList&, const FOSTimeInter-val&, const FoEcTime&, const EcTInt&)`

Request FmSmSpacecraft to map the command list into an ATC buffer model. Instantiates an FmMsATCMapRequest object to be sent to FmSmSpacecraft.

`RWSlistCollectables&` **`MapLateChange`**`(const FmMnDirectiveList&, const FOSTimeInterval&, const FoEcTime&, const EcTInt&)`

Requests FmSmSpacecraft to map the late change command list into the correct buffer model. Instantiates an FmMsATCMapRequest object to be sent to FmSmSpacecraft.

`RWSlistCollectables&` **`Receive`**`(EcTVoid)`

Receives the response from FmSmSpacecraftModel It receives either A list of FmMsLoadData objects or a FmMsATCBufferInfo object

`EcTVoid` **`Send`**`(const RWCollectable&)`

Sends messages to FmSmSpacecraftModel. Sends FmMsATCMapRequest, FmMsDeleteATCBuffers, or FmMsUpdateBuffer.

`EcTVoid` **`UpdateBuffer`**`(const FmMsUpdateBuffer&)`

Request the buffer be updated to a new status

FmMsLoadData

`class` **`FmMsLoadData`**

This class is sent to CMS Schedule controller. From this class the ATC load directives are used to create the ATC binary uplink load. If the DAS needs to be partitioned multiple FmMsLoadData objects are returned to CMS Schedule Controller.

Private Data

`EcTInt` **`myDirListAddr`**

This is the next directive in the processing list. If the list is completely processed the is set to NULL. If the list requires further processing, that is the DAS/ ATC load need to be partitioned, it is set to the next directive in the list. This is where the partitioned load needs to begin.

`FmMnDirectiveList` **`myDirectiveList`**

This is the portion of the DAS/ATC directive list being currently processed that will be used to create the ATC binary uplink load It may be all of the DAS or part of the DAS if the ATC buffer cannot hold all of the commands - that is the DAS is being partitioned

`RWCString` **`myLoadName`**

This is the load name create by ATC buffer model

`FOSTimeInterval` **`myUplinkWindow`**

This is the uplink window for the ATC uplink directive list

FoLiATCLoad

`class` **`FoLiATCLoad`**

class definition

Base Classes

`public` **`FoLiLoad`**

Public Functions

`EcTInt` **`BuildUplinkLoad`**`()`

Builds the uplinkable load and the load report.

`EcTInt` **`ComposeReport`**`()`

Fills in the attributes of the report and stores it.

`FoMsCMSStatus&` **`CreateLoad`**`(const FmMnDirectiveList&)`

Populates the load object and its aggregate parts.

Private Data

`FmMnDirectiveList` **`myCriticalCommands`**

The critical commands in the load.

`EcTInt` **`myCriticalFlag`**

An indicator of critical commands existing in the load.

`EcTInt` **`myDASId`**

The DAS id for which the load is generated.

`FmMnDirectiveList` **`myDirectiveList`**

The entire list of directives that constitute the load.

`FoLiATCLoadReport*` **`myLoadReport`**

A pointer to the load report for this load.

FoLiATCLoadReport

`class` **`FoLiATCLoadReport`**

class definition

Base Classes

`public` **`FoLiLoadReport`**

Private Data

`FmMnDirectiveList` **`myCommandList`**

The entire list of commands which constitute the load.

`FmMnDirectiveList` **`myControlCommands`**

The control commands contained in the load.

`RWTime` **`myStartTime`**

The time of the first command in the load.

`RWTime` **`myStopTime`**

the time of the last command in the load.

FoLiLoad

`class` **`FoLiLoad`**

Base Classes

`public` **`RWCollectable`**

Public Functions

`virtual EcTInt` **`BuildUplinkLoad`**`(const FoLiLoadImage&)`

Builds the uplinkable load and the load report.

`FoMsCMSStatus&` **`CreateLoad`**`(const FoMsLoadGenReq&)`

Reads in the file from the request and creates the load.

`EcTInt` **`GenerateLoadImage`**`(const FoLiLoadContents&)`

Generates the binary for the load and stores it in a file.

Private Data

`RWCString` **`myDestination`**

The destination on the spacecraft for the load.

`RWCString` **`myDirectory`**

The directory where the load contents file from which the load is generated exists.

`FoLiLoadContents` **`myLoadContents`**

The load contents object.

`RWCString` **`myLoadName`**

The name of the load.

`EcTInt` **`myLoadSize`**

The size of the load in bytes.

`EcTInt` **`myNumberOfPieces`**

The number of uplink loads for this load.

`RWCString` **`myOwner`**

The id of the owner of the load.

`EcTInt` **`mySizeOfLastPiece`**

The number of bytes of the last uplinkable load.

`EcTInt` **`mySpacecraftId`**

The id of the spacecraft for which the load is valid.

`FoMsCMSStatus` **`myStatus`**

The processing status of the load.

`RWSlistCollectables` **`myUplinkLoads`**

The uplinkable portions of the load.

`FOSTimeInterval` **`myUplinkPeriod`**

The uplink period of the load.

FoLiLoadReport

`class` **`FoLiLoadReport`**

class definition

Base Classes

`public` **`FoDsFile`**

Private Data

`EcTInt` **`myEndLocation`**

The last memory location used by the load.

`RWCString` **`myLoadName`**

The name of the load for which this report was written.

`EcTInt` **`mySize`**

The size of the load in bytes.

`EcTInt` **`myStartLocation`**

The first memory location used by the load.

`RWCString` **`myType`**

the type of the load for which this report was written.

`FOSTimeInterval` **`myUplinkPeriod`**

The uplink window of the load for which this report was written.

FoLiUplinkLoad

`class` **`FoLiUplinkLoad`**

class definition

Base Classes

`public` **`FoDsFile`**

Public Functions

`EcTInt` **`BuildLoad`**`(const FoLiLoadImage&)`

Generates the CRC for the load, puts the load into packets, and stores the load with DMS.

`EcTInt*` **`BuildLoadData`**`()`

Sets the command destination information and fills in the command data.

`EcTInt*` **`CCSDSWrap`**`()`

Generates the packets for the load.

FoScActivity

`class` **`FoScActivity`**

Defines one activity in the Activity List

Private Data

`RWString` **`myActDefName`**

Name of the Activity to be expanded

`EcTInt` **`myActId`**

Activity ID

`RWSlistCollectables` **`myParamValueList`**

Parameter values to be substituted into the appropriate commands

`RWTime` **`myStartTime`**

Start time of the Activity

`RWTime` **`myStopTime`**

Stop time of the Activity

FoScActivityList

`class` **`FoScActivityList`**

Activity List class received from PAS

Private Data

`EcTInt` **`myId`**

Activity list ID (DASid)

FoScDetActSched

`class` **`FoScDetActSched`**

Defines a Detailed Activity Schedule (DAS)

Base Classes

`public` **`FoScActivityList`**

Protected Data

`RWSlistCollectables` **`myOrbitalEvent`**

List of orbital events associated with the input activity list

`RWTime` **`myStartTime`**

Start time of the Activity List

`RWTime` **`myStopTime`**

Stop time of the Activity List

`FOSTimeInterval` **`myUplinkWindowReq`**

Requested uplink window for the load to be generated from the Activity List

`RWString` **`myVersion`**

Version number

FoScLateChange

`class` **`FoScLateChange`**

Defines a Late Change Activity List

Base Classes

`public` **`FoScDetActSched`**

Private Data

`RWTime` **`myStartTime`**

Start time of the Late Change Activity List

FoScOrbitalEvents

`class` **`FoScOrbitalEvents`**

Defines each orbital event associated with a DAS

Private Data

`RWString` **`myName`**

Name of the orbital event

`RWTime` **`myTime`**

Time of the orbital event

FoScSimulationSched

`class` **`FoScSimulationSched`**

Defines a simulation Activity List

Base Classes

`public` **`FoScActivityList`**

Private Data

`EcTInt` **`myATCBufferStart`**

ATC buffer start time to be used when partitioning the load

`RWTime` **`myStartTime`**

Start time of the simulation Activity List

`RWTime` **`myStopTime`**

Stop Time of the Activity List

`RWString` **`myVersion`**

Version number of the Activity List

FoScUplinkActivity

`class` **`FoScUplinkActivity`**

stp/omt class definition 3290603

Base Classes

`public` **`FoScActivity`**

Protected Data

`RWString` **`myLoadName`**

Load name for the associated uplink request activity list

FoEcComment

`class` **`FoEcComment`**

Defines a comment directive

Base Classes

`public` **`FoEcGroundDirective`**

Private Data

`RWCString` **`myKeyword`**

Keyword indicating directive is a comment

FoEcDeltaTime

`class` **`FoEcDeltaTime`**

Defines Delta time

Base Classes

`public` **`FoEcTime`**

Delta Time

Private Data

`EcTChar` **`myPlusMinusSign`**

Indicates whether the delta time is positive or negative. Used for computing absolute time.

`EcTChar` **`myStartStopIndicator`**

Indicates whether the delta time is associated with the start or stop time of the activity

FoEcDirective

`class` **`FoEcDirective`**

Defines an individual directive

Public Functions

`void` **`CheckSyntax`**`(EcTInt errcode)`

Checks to ensure the syntax of the directive is valid

`void` **`Execute`**`(void)`

`void` **`LogDirective`**`(void)`

Logs the directive

`void` **`Parse`**`(void)`

Parses the directive

`void` **`UpdateStatus`**`(void)`

updates the status of the directive

Private Data

`EcTInt` **`myActivityId`**

Id of the activity that this directive was expanded from

`EcTInt` **`myDASId`**

id of the DAS that this directive was expanded from

`FuTdDataSource*` **`myDataSourceId`**

id of the originator of the directive

`RWCString` **`myDirectiveText`**

text describing this directive

`FuGsGroundScriptControl*` **`myGndScript`**

ground script for the associated directive

`EcTInt` **`myLineNum`**

line number from the associated expansion

`RWSlistCollectables` **`myParameters`**

list of parameters associated with this directive

`FoClProcedure*` **`myProc`**

indicates which procedure this directive was expanded from

`FuClProcControlWin*` **`myProcControl`**

`enum` **`myProcFlag`**

indicates whether this directive was expanded from a procedure enum **mySource**

source of the directive

EcTInt **myStatus**

status of the directive

Private Types

enum

Enumerators

gs
manual
proc

enum

Enumerators

n
y

FoEcGroundDirective

class **FoEcGroundDirective**

Defines an individual ground directive

Base Classes

public **FoEcDirective**

Private Data

RWCString **myKeyword**

Defines the keyword for the ground directive

FoEcLabel

class **FoEcLabel**

Base Classes

public **FoEcGroundDirective**

Private Data

RWCString **myName**

Name of the label associated with the ground directive

EcTInt **myOffset**

FoEcRTCommand

class **FoEcRTCommand**

Defines a real time command directive

Base Classes

`public` **`FoEcGroundDirective`**

Private Data

`RWBitVec` **`myBinary`**

Binary data associated with the real time command

`RWCString` **`myMnemonic`**

Mnemonic which defines the real time command

FoEcSpaceDirective

`class` **`FoEcSpaceDirective`**

Defines a spacecraft directive

Base Classes

`public` **`FoEcDirective`**

Public Functions

`EcTInt` **`FigureBinary`**`()`

Generates the binary representation of this directive.

Private Data

`RWBitVec` **`myBinary`**

The binary representation of this directive, the format of which is described by the ICD.

`EcTInt` **`myInhibitId`**

The group id indicating what resource this directive could have an effect on.

`RWCString` **`myMnemonic`**

The mnemonic for the directive.

`EcTInt` **`myRTSFlag`**

An indicator specifying if this directive is part of an RTS or not.

FoEcSpaceTime

`class` **`FoEcSpaceTime`**

class definition

Base Classes

`public` **`FoEcTime`**

Public Functions

`EcTFloat` **`GetConversionFactor`**`()`

Returns the conversion factor.

`EcTVoid` **`SetConversionFactor`**`(EcTFloat)`

Sets the conversion factor to the input value.

Private Data

`EcTFloat` **`myConversionFactor`**

The numeric factor which converts actual seconds to spacecraft seconds.

FoEcTime

`class` **`FoEcTime`**

class definition

Base Classes

`public` **`RWTime`**

Public Functions

`RWTime&` **`GetEpoch`**`()`

Returns the epoch.

`EcTVoid` **`SetEpoch`**`(const RWTime&)`

Sets the epoch to the input time.

Private Data

`RWTime` **`myEpoch`**

The epoch upon which the time is based. Time is computed as the number of seconds since the epoch. The default epoch for RWTime is Jan. 1, 1901 at 00:00:00.

3.3 Ground Schedule

The Ground Schedule is a persistent process that runs on the FOS Data Server. It maintains a continuous schedule of ECL directives, including planned real-time commands, ground configuration directives, and comments representing spacecraft stored commands and orbital events. Lists of directives to be added to the continuous schedule are sent to Ground Schedule by Schedule Controller. In response to a late change request from PAS, Schedule Controller may also request the deletion of directives with a particular DAS id from the continuous schedule.

Ground Schedule uses its continuous list of directives to generate responses to various requests. In response to a constraint check list request, Ground Schedule provides a list of directives for a specified time range to Schedule Controller. In response to a ground script request from FUI, Ground Schedule generates a ground script, which is a time ordered list of directives, for the requested time range and stores it with DMS. In response to an expected state request, Ground Schedule generates an expected state table for a specified time and returns it to telemetry. An expected state table is based on scheduled stored and real-time commands and consists of a list of telemetry parameter ids and their predicted values.

Ground Schedule is also responsible for generating the integrated report. This report is generated automatically whenever a DAS is processed and includes all scheduled ECL directives and orbital events for the time span of the previous Detailed Activity Schedule.

3.3.1 Ground Schedule Context

Figure 3.3-1 shows the context diagram for the Ground Schedule. The Ground Schedule has four interfaces.

CMS Schedule Controller

- Ground Schedule receives a directive list to be merged into the existing schedule.
- Ground Schedule receives a request to delete certain directives from the schedule.
- Ground Schedule receives a request to return a list of certain types of directives.
- Ground Schedule sends a directive list in response to a list request.

DMS

- Ground Schedule receives the Command Execution Verification (CEV) definitions upon initialization.

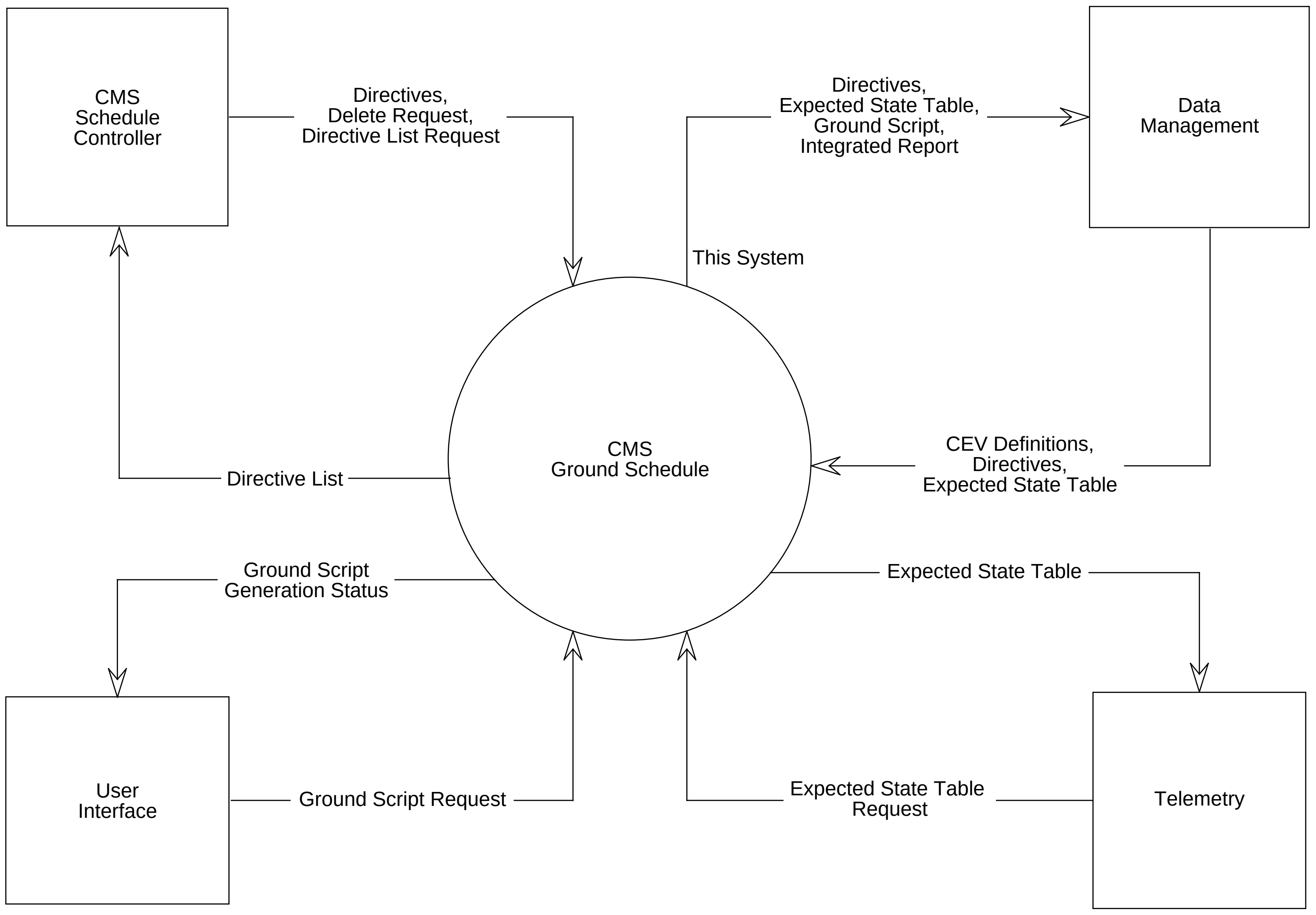

CMS Schedule Controller
Directives, Delete Request, Directive List Request
Directives, Expected State Table, Ground Script, Integrated Report
Data Management
This System
CMS Ground Schedule
Directive List
CEV Definitions, Directives, Expected State Table
Ground Script Generation Status
Expected State Table
User Interface
Ground Script Request
Expected State Table Request
Telemetry

Figure 3.3-1. Ground Schedule Context Diagram

- Ground Schedule receives its schedule of directives in a file upon initialization.
- Ground Schedule receives an Expected State Table file upon initialization.
- Ground Schedule stores its schedule of directives upon a schedule change (addition or deletion).
- Ground Schedule stores an Expected State Table whenever one is generated.
- Ground Schedule stores a Ground Script whenever one is generated.
- Ground Schedule stores an Integrated Report whenever one is generated.

Telemetry

- Ground Schedule receives a request to generate an Expected State Table.
- Ground Schedule sends an Expected State Table when one is generated upon request.

FUI

- Ground Schedule receives a request to generate a Ground Script.
- Ground Schedule sends a status indicating the storage location of the Ground Script.

3.3.2 Ground Schedule Interfaces

Table 3.3.2. Ground Schedule Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Delete Directives	FmGsGroundData	Proxy between CMS:Schedule Controller and CMS: Ground Schedule. Requests deletion of directives from the Ground Schedule	CMS: Ground Schedule	CMS: Schedule Controller	1/month
Add Directives	FmGsGroundData	Proxy between CMS:Schedule Controller and CMS: Ground Schedule.	CMS: Ground Schedule	CMS: Schedule Controller	1/day
	FmMnDirectiveList	List of directives to add to the Ground Schedule.			
Return Directive List	FmGsGroundData	Proxy between CMS:Schedule Controller and CMS: Ground Schedule.	CMS: Ground Schedule	CMS: Schedule Controller	1/day
	FmGsListRequest	Contains list of DAS id's and a time. All directives in the Ground Schedule that occur after the time and that have DAS Id's that are in the list will be returned.			
	FmMnDirectiveList	List of directives.			
Generate Ground Script	FmMsGenerateOpAids	Proxy between CMS:Ground Schedule and FUI	CMS: Ground Schedule	FUI: Ground Script Controller	3/day
	FoMsGsGenReq	Request for Ground Script to be created for given times.			
	FoGsStatus	Results of Ground Script Generation.			
Generate Expected State Table	FmMsExpectedStateTable	Proxy between Ground Schedule and TLM.	CMS: Ground Schedule	TLM: FtTlSCStateCheck	1/contact
	FoMsTableRequest	Request that a table be generated for a given time.			
	FoTlExpectedState	List of FoTlExpectedValue objects.			
	FoTlExpectedValue	Expected value for a particular parameter id.			

3.3.3 Ground Schedule Object Model

Figure 3.3-2 shows the first page of the object model for the Ground Schedule. Ground Schedule is an independent process which maintains the scheduled directives. Figure 3.3-3 shows the class FmMnDirectiveList and the entire FoEcDirective structure. FmScGroundSchedule is responsible for adding directives to and deleting directives from its list and for generating the Ground Script, Expected State Table, and Integrated Report based on the directive list. It has four interfaces that are used to allow other process access to the schedule. FmScGroundSchedule generates an FoGsGroundScript, which is derived from FoDsFile. Figure 3.3-4 shows the classes which are derived from FoDsFile. FoGsCEVTable is generated by the Ground Schedule upon initialization using information obtained from DMS. The CEV Table contains many FoGsCEVDataField objects. Each Data Field contains information about one CEV pid. The CEV Table is used to generated an Expected State Table.

FoTlExpectedState is the Expected State Table generated using the CEV Table. FoTlExpectedState consists of one to many FoTlExpectedValue objects, each of which contain information about one parameter id. The FoTlExpectedState is updated based on a FoMsTableRequest received from the proxy FmMsExpectedStateTable. This proxy is how TLM requests the Ground Schedule to generate an Expected State Table for a certain time.

FoRpIntegratedReport is a file object that is generated whenever an expanded Detailed Activity Schedule is received from the Schedule Controller through the proxy FmGsGroundData. This proxy also sends a FmGsListRequest to the Ground Schedule, requesting a portion of the schedule be returned to the Schedule Controller.

The class FmMsGenerateOpAids is an interface proxy with FUI. It receives a FoMsGsGenReq object from FUI, which is used to request that a Ground Script covering a certain period of time be generated. Ground Schedule sends to the proxy a FoGsStatus, indicating the storage location of the Ground Script and a brief status message.

3.3.4 Ground Schedule Dynamic Model

The Ground Schedule dynamic model consists of the following scenarios:

- Ground Schedule Initialization
- Expected State Table Generation

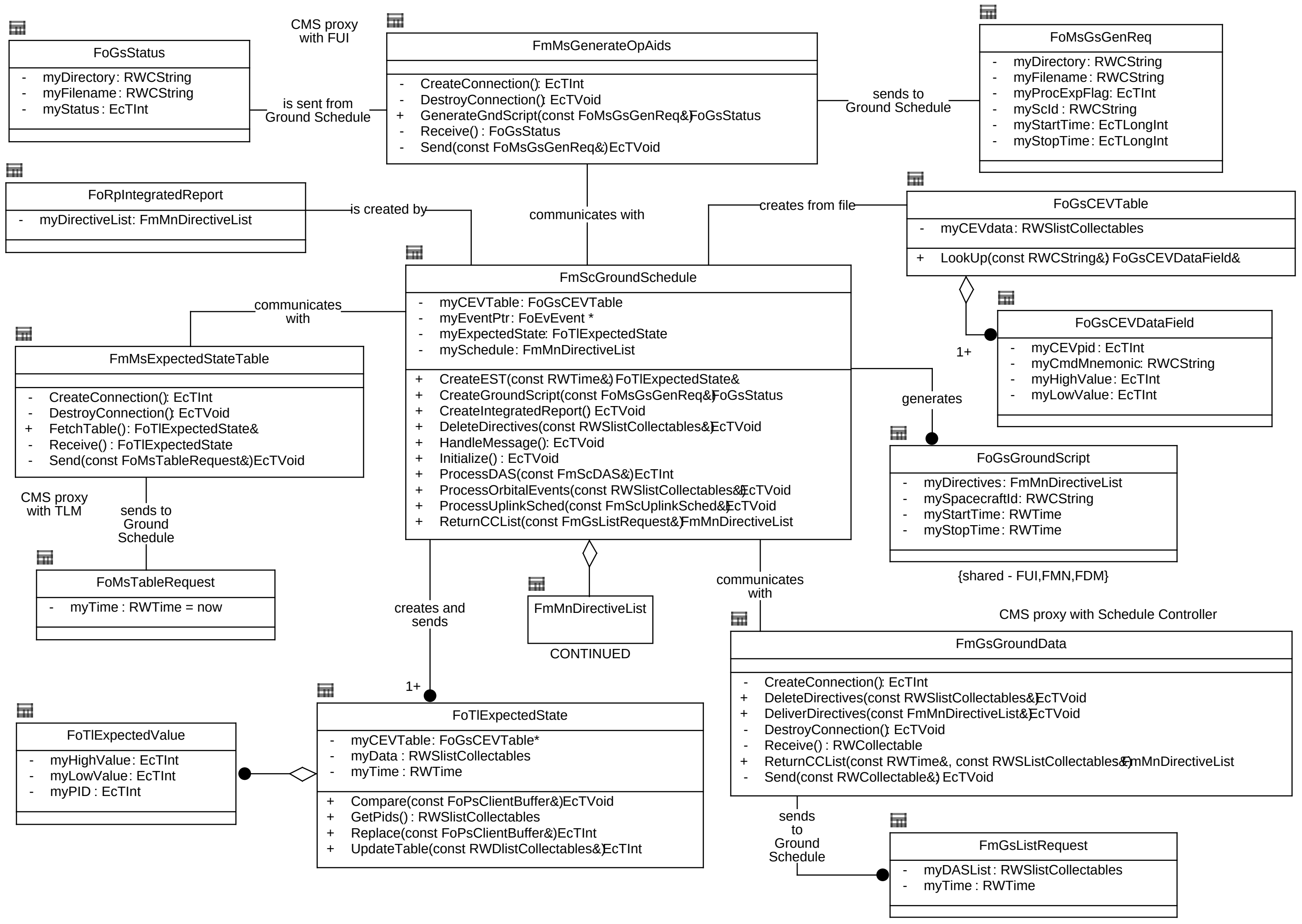

CMS proxy with FUI
FmMsGenerateOpAids
- CreateConnection(): EcTInt
- DestroyConnection(): EcTVoid
+ GenerateGndScript(const FoMsGsGenReq&):FoGsStatus
- Receive() : FoGsStatus
- Send(const FoMsGsGenReq&):EcTVoid
FoGsStatus
- myDirectory : RWCString
- myFilename : RWCString
- myStatus : EcTInt
is sent from Ground Schedule
sends to Ground Schedule
FoMsGsGenReq
- myDirectory : RWCString
- myFilename : RWCString
- myProcExpFlag: EcTInt
- myScId : RWCString
- myStartTime : EcTLongInt
- myStopTime : EcTLongInt
FoRpIntegratedReport
- myDirectiveList: FmMnDirectiveList
is created by
communicates with
creates from file
FoGsCEVTable
- myCEVdata: RWSlistCollectables
+ LookUp(const RWCString&): FoGsCEVDataField&
FmScGroundSchedule
- myCEVTable : FoGsCEVTable
- myEventPtr : FoEvEvent *
- myExpectedState: FoTlExpectedState
- mySchedule: FmMnDirectiveList
+ CreateEST(const RWTime&): FoTlExpectedState&
+ CreateGroundScript(const FoMsGsGenReq&):FoGsStatus
+ CreateIntegratedReport() EcTVoid
+ DeleteDirectives(const RWSlistCollectables&):EcTVoid
+ HandleMessage(): EcTVoid
+ Initialize() : EcTVoid
+ ProcessDAS(const FmScDAS&):EcTInt
+ ProcessOrbitalEvents(const RWSlistCollectables&):EcTVoid
+ ProcessUplinkSched(const FmScUplinkSched&):EcTVoid
+ ReturnCCList(const FmGsListRequest&):FmMnDirectiveList
communicates with
FmMsExpectedStateTable
- CreateConnection(): EcTInt
- DestroyConnection(): EcTVoid
+ FetchTable(): FoTlExpectedState&
- Receive() : FoTlExpectedState
- Send(const FoMsTableRequest&):EcTVoid
CMS proxy with TLM
sends to Ground Schedule
FoMsTableRequest
- myTime : RWTime = now
1+
FoGsCEVDataField
- myCEVpid : EcTInt
- myCmdMnemonic: RWCString
- myHighValue : EcTInt
- myLowValue : EcTInt
generates
FoGsGroundScript
- myDirectives : FmMnDirectiveList
- mySpacecraftId: RWCString
- myStartTime : RWTime
- myStopTime : RWTime
{shared - FUI,FMN,FDM}
creates and sends
FmMnDirectiveList
CONTINUED
communicates with
CMS proxy with Schedule Controller
FmGsGroundData
- CreateConnection(): EcTInt
+ DeleteDirectives(const RWSlistCollectables&):EcTVoid
+ DeliverDirectives(const FmMnDirectiveList&):EcTVoid
- DestroyConnection(): EcTVoid
- Receive() : RWCollectable
+ ReturnCCList(const RWTime&, const RWSListCollectables&):FmMnDirectiveList
- Send(const RWCollectable&): EcTVoid
1+
FoTlExpectedValue
- myHighValue : EcTInt
- myLowValue : EcTInt
- myPID : EcTInt
FoTlExpectedState
- myCEVTable: FoGsCEVTable*
- myData : RWSlistCollectables
- myTime : RWTime
+ Compare(const FoPsClientBuffer&):EcTVoid
+ GetPids() : RWSlistCollectables
+ Replace(const FoPsClientBuffer&):EcTInt
+ UpdateTable(const RWDlistCollectables&):EcTInt
sends to Ground Schedule
FmGsListRequest
- myDASList : RWSlistCollectables
- myTime : RWTime

Figure 3.3-2. Ground Schedule Object Model - page 1

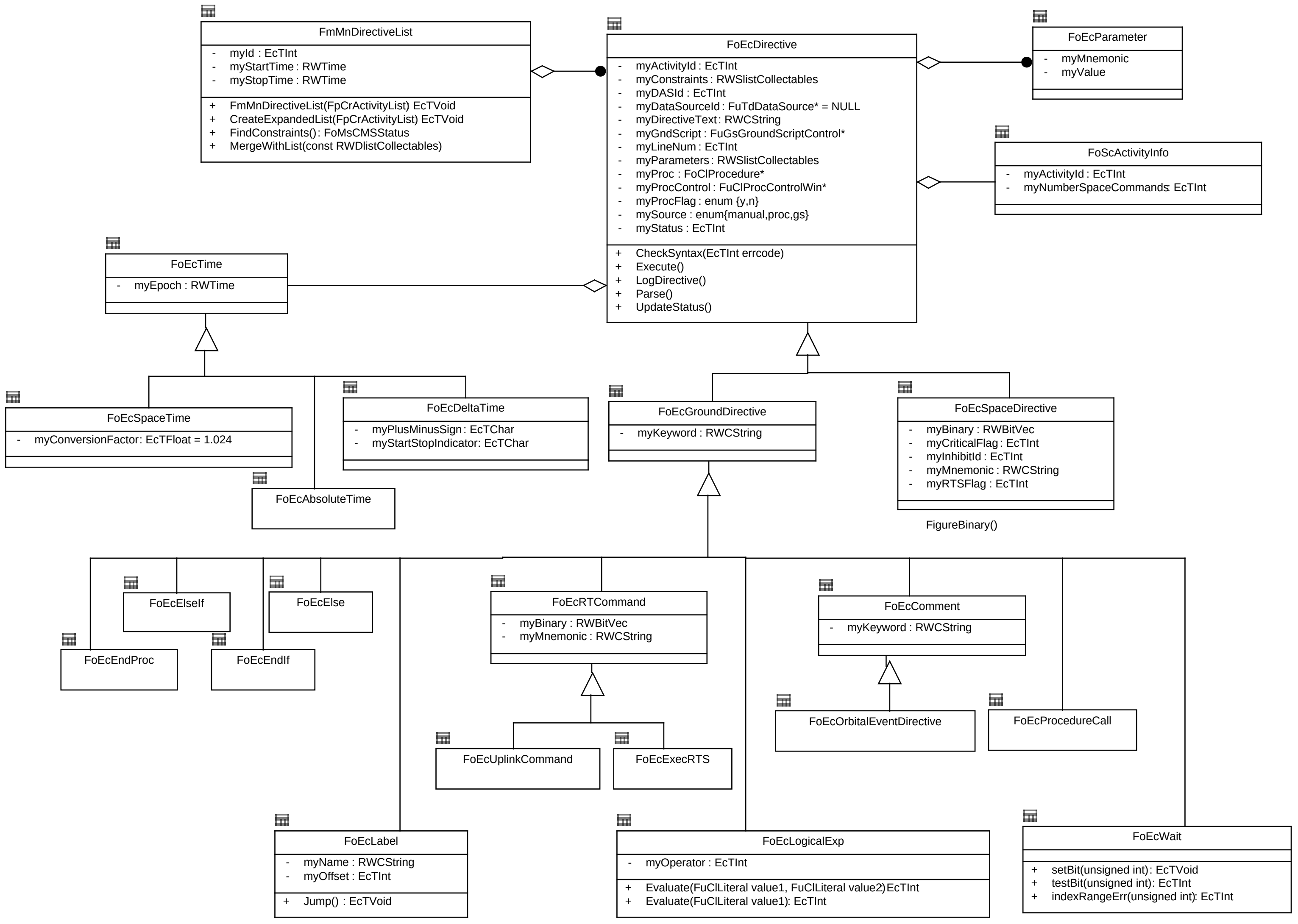

FmMnDirectiveList
- myId : EcTInt
- myStartTime : RWTime
- myStopTime : RWTime
+ FmMnDirectiveList(FpCrActivityList) EcTVoid
+ CreateExpandedList(FpCrActivityList) EcTVoid
+ FindConstraints(): FoMsCMSStatus
+ MergeWithList(const RWDlistCollectables)
FoEcDirective
- myActivityId : EcTInt
- myConstraints : RWSlistCollectables
- myDASId : EcTInt
- myDataSourceId : FuTdDataSource* = NULL
- myDirectiveText : RWCString
- myGndScript : FuGsGroundScriptControl*
- myLineNum : EcTInt
- myParameters : RWSlistCollectables
- myProc : FoClProcedure*
- myProcControl : FuClProcControlWin*
- myProcFlag : enum {y,n}
- mySource : enum{manual,proc,gs}
- myStatus : EcTInt
+ CheckSyntax(EcTInt errcode)
+ Execute()
+ LogDirective()
+ Parse()
+ UpdateStatus()
FoEcParameter
- myMnemonic
- myValue
FoScActivityInfo
- myActivityId : EcTInt
- myNumberSpaceCommands: EcTInt
FoEcTime
- myEpoch : RWTime
FoEcSpaceTime
- myConversionFactor: EcTFloat = 1.024
FoEcDeltaTime
- myPlusMinusSign : EcTChar
- myStartStopIndicator: EcTChar
FoEcAbsoluteTime
FoEcGroundDirective
- myKeyword : RWCString
FoEcSpaceDirective
- myBinary : RWBitVec
- myCriticalFlag : EcTInt
- myInhibitId : EcTInt
- myMnemonic : RWCString
- myRTSFlag : EcTInt
FigureBinary()
FoEcElseIf
FoEcElse
FoEcEndProc
FoEcEndIf
FoEcRTCommand
- myBinary : RWBitVec
- myMnemonic : RWCString
FoEcComment
- myKeyword : RWCString
FoEcUplinkCommand
FoEcExecRTS
FoEcOrbitalEventDirective
FoEcProcedureCall
FoEcLabel
- myName : RWCString
- myOffset : EcTInt
+ Jump() : EcTVoid
FoEcLogicalExp
- myOperator : EcTInt
+ Evaluate(FuClLiteral value1, FuClLiteral value2)EcTInt
+ Evaluate(FuClLiteral value1): EcTInt
FoEcWait
+ setBit(unsigned int): EcTVoid
+ testBit(unsigned int): EcTInt
+ indexRangeErr(unsigned int): EcTInt

Figure 3.3-3. Ground Schedule Object Model - page 2 - Directives

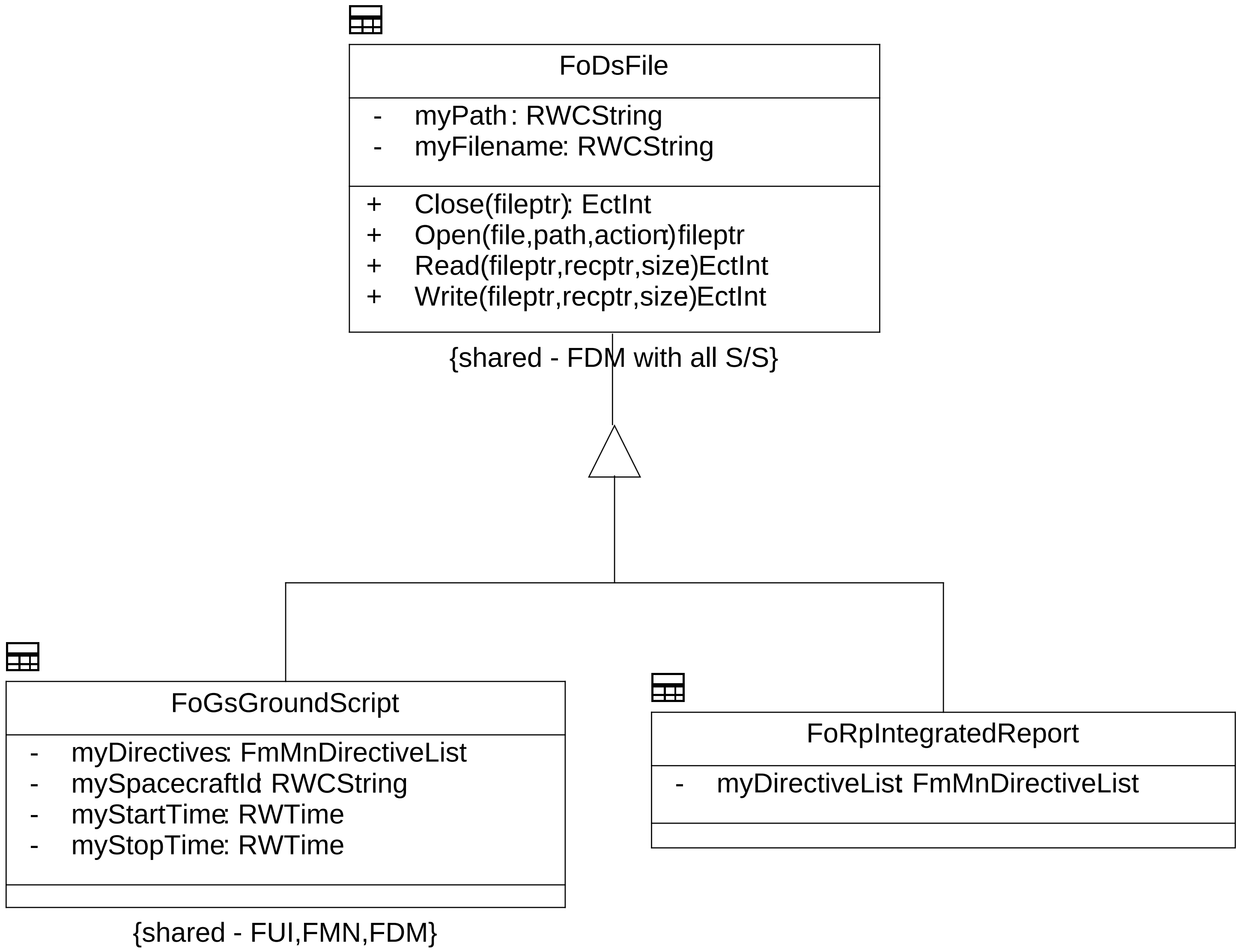

FoDsFile
- myPath : RWCString
- myFilename: RWCString
+ Close(fileptr): EctInt
+ Open(file,path,action) fileptr
+ Read(fileptr,recptr,size) EctInt
+ Write(fileptr,recptr,size) EctInt
{shared - FDM with all S/S}
FoGsGroundScript
- myDirectives: FmMnDirectiveList
- mySpacecraftId RWCString
- myStartTime: RWTime
- myStopTime: RWTime
{shared - FUI,FMN,FDM}
FoRpIntegratedReport
- myDirectiveList FmMnDirectiveList

Figure 3.3-4 . Ground Schedule Object Model - Files

- DAS Processing
- Ground Script Generation

3.3.4.1 Ground Schedule Initialization Scenario

3.3.4.1.1 Ground Schedule Initialization Abstract

The Initialization Scenario describes how the Ground Schedule process is initialized. The Ground Schedule ingests the information it needs from DMS.

3.3.4.1.2 Ground Schedule Initialization Summary Information

Interfaces:

- DMS

Stimulus:

- Ground Schedule process is started.

Desired Response:

- Schedule is restored from DMS
- Expected State Table is restored from DMS
- CEV Table is restored from DMS

Pre-Conditions:

- DMS is accessible

Post-Conditions:

- Ground Schedule is ready to process requests.

3.3.4.1.3 Ground Schedule Initialization Description

Figure 3.3-5 shows the Ground Schedule Initialization event trace. When the initialize function is called, Ground Schedule requests from DMS the data file containing the schedule. The data file is returned, and Ground Schedule recreates its schedule from the information in the file. Ground Schedule requests from DMS the data file containing the CEV Table. The data file is returned, and Ground Schedule creates its CEV Table from the information in the file. Ground Schedule requests from DMS the data file containing the Expected State Table. The data file is returned, and Ground Schedule creates its Expected State from the information in the file.

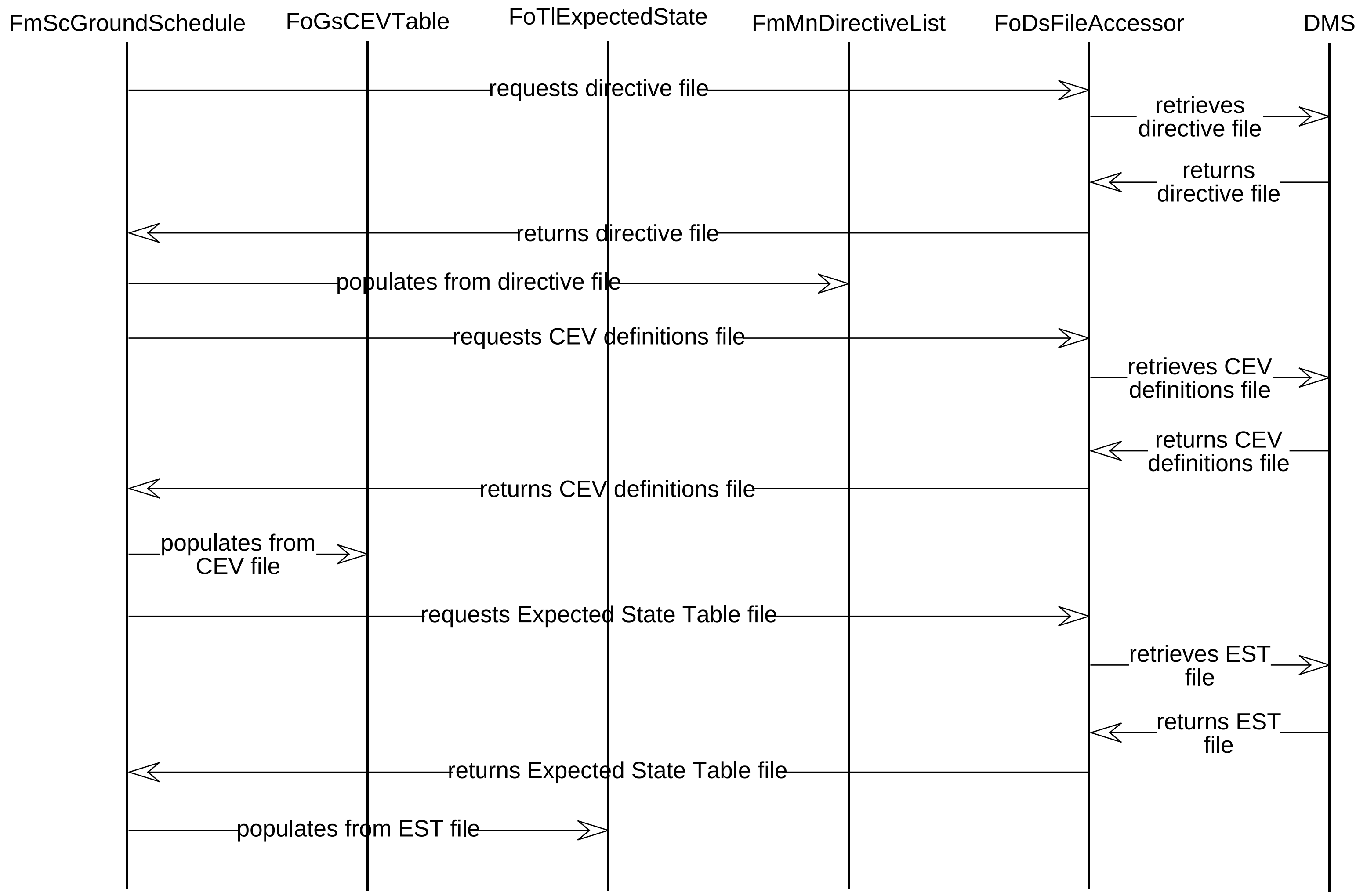


Figure 3.3-5. Ground Schedule Initialization Event Trace

3.3.4.2 Expanded DAS Processing Scenario

3.3.4.2.1 Expanded DAS Processing Abstract

The Expanded DAS processing scenario describes how a Detailed Activity Schedule is processed in the Ground Schedule when one is received from the Schedule Controller.

3.3.4.2.2 Expanded DAS Processing Summary Information

Interfaces:

- Schedule Controller
- DMS

Stimulus:

- Receipt of a directive list from Schedule Controller

Desired Response:

- Directives merged into existing schedule
- Integrated Report generated and stored with DMS

Pre-Conditions:

- Ground Schedule software has been initialized

Post-Conditions:

- Schedule checkpointed with DMS
- The Ground Schedule's most recent Expected State Table is updated

3.3.4.2.3 Expanded DAS Processing Description

Figure 3.3-6 shows the Expanded DAS Processing Event Trace. The Ground Schedule receives a directive list from the internal interface proxy FmGsGroundData. This is a proxy with Schedule Controller. The Ground Schedule adds the directives in the directive list to its existing schedule. It then generates an Integrated Report for the previous DAS received and stores it with DMS. Ground Schedule creates an Expected State Table based on the current time and stores it with DMS. Finally, the Ground Schedule stores its schedule with DMS.

3.3.4.3 Delete from Schedule Scenario

3.3.4.3. Delete from Schedule Abstract

The Delete from Schedule scenario describes how directives are deleted from the Ground Schedule. Directives need to be deleted whenever a Late Change Detailed Activity Schedule is received from Planning and Scheduling.

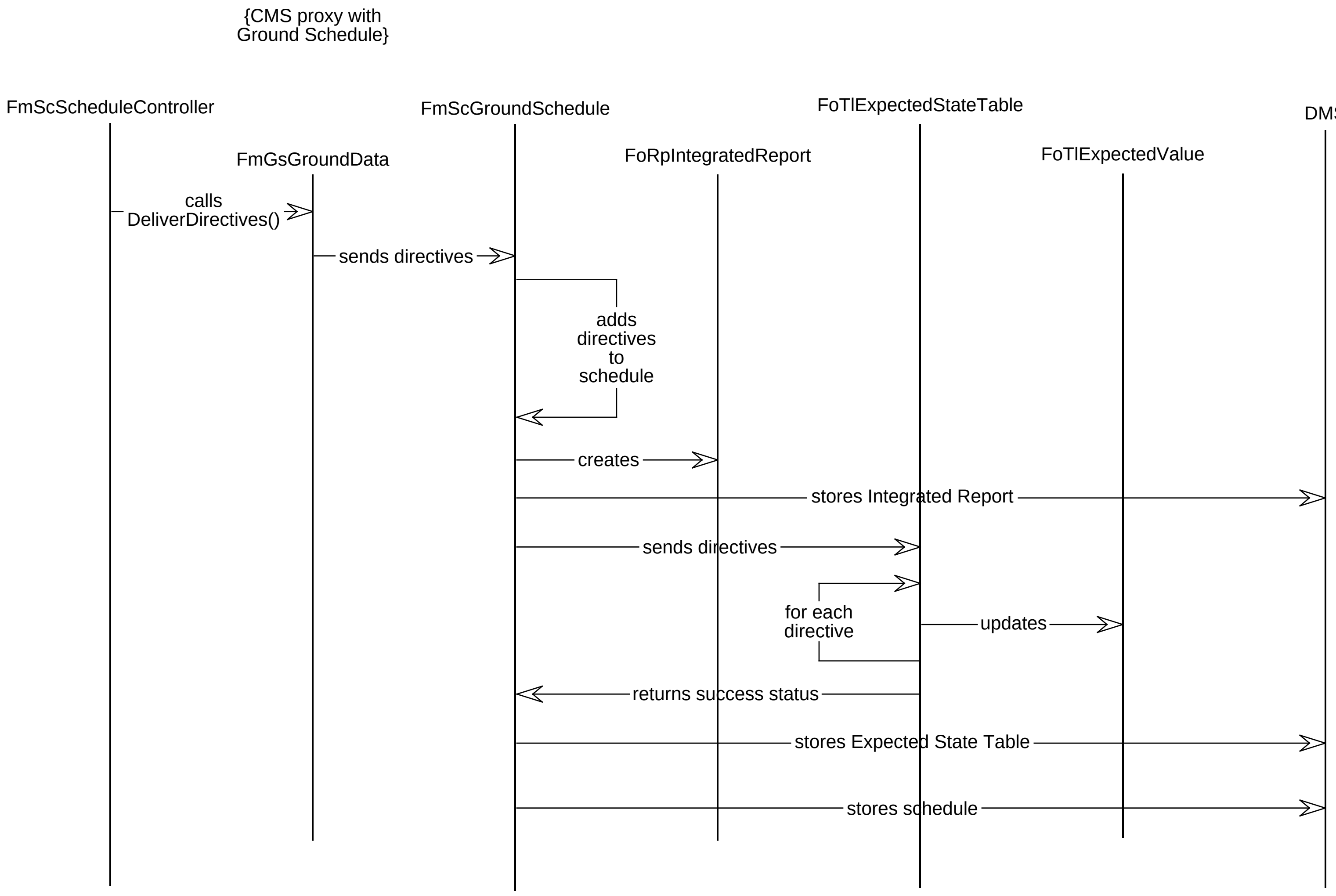


Figure 3.3-6. Expanded DAS Processing Event Trace

3.3.4.3.2 Delete from Schedule Summary Information

Interfaces:

- Schedule Controller

Stimulus:

- Receipt of a delete request from the Schedule Controller

Desired Response:

- Directives specified in request deleted from Ground Schedule

Pre-Conditions:

- Ground Schedule software is initialized
- Ground Schedule is populated with directives

Post-Conditions:

- Schedule is checkpointed to DMS

3.3.4.3.3 Delete from Schedule Description

Figure 3.3-7 shows the Delete from Schedule event trace. A delete request is received from Schedule controller through the proxy FmMsGroundData. Ground Schedule determines which directives should be deleted based on the DAS Id's in the request. All directives having a DAS Id which is listed in the request are deleted. Ground Schedule stores the schedule with DMS as a checkpoint file.

3.3.4.4 Expected State Table Generation Scenario

3.3.4.4.1 Expected State Table Generation Abstract

The Expected State Table Generation scenario describes the generation of an Expected State Table upon request from the Telemetry subsystem.

3.3.4.4.2 Expected State Table Generation Summary Information

Interfaces:

- TLM
- DMS

Stimulus:

- Receipt of a FoMsTableRequest from TLM

Desired Response:

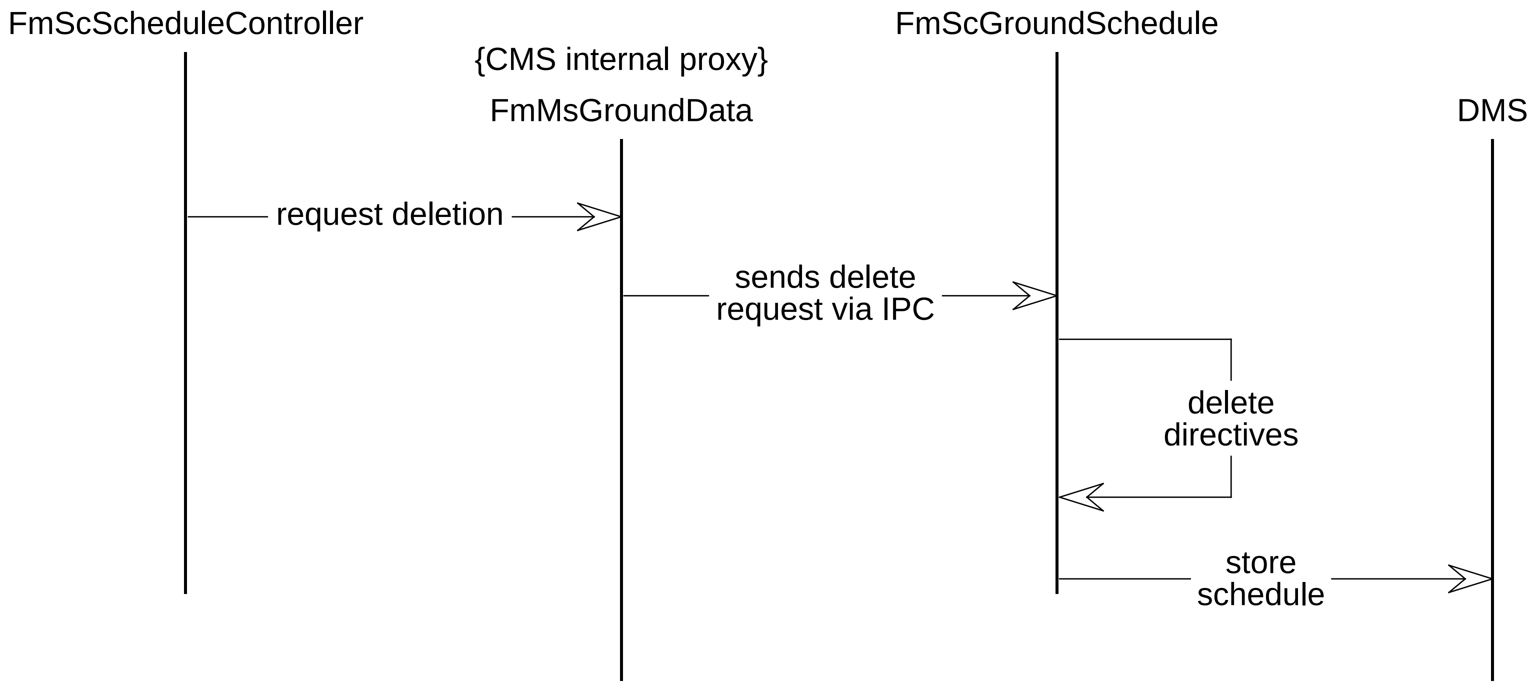


Figure 3.3-7. Delete from Schedule Event Trace

- Expected State Table sent to TLM

Pre-Conditions:

- Ground Schedule software has been initialized
- The schedule has been populated with directives
- A previously generated Expected State Table exists

Post-Conditions:

- The Ground Schedule's most recent Expected State Table has been updated

3.3.4.4.3 Expected State Table Generation Scenario Description

Figure 3.3-8 shows the Expected State Table Generation Event Trace. The Ground Schedule receives a FoMsTableRequest from the interface proxy class FmMsExpectedStateTable. Ground Schedule uses the time in the request to create an FoTlExpectedState object based on a previously generated table. Ground Schedule determines which directives from its schedule will effect the table and passes them to the FoTlExpectedState. For each directive passed in, FoTlExpectedState looks up the command mnemonic in its CEV Table and updates the appropriate FoTlExpectedValue. When all of the directives have been looked up, Ground Schedule sends the Expected State table to the interface proxy. Finally, Ground Schedule stores the Expected State table with DMS.

3.3.4.5 Ground Script Generation Scenario

3.3.4.5.1 Ground Script Generation Abstract

The Ground Script Generation scenario describes the generation of a Ground Script based on a request sent to the Ground Schedule from FUI.

3.3.4.5.2 Ground Script Generation Summary Information

Interfaces:

- FUI
- DMS

Stimulus:

- Receipt of an FoMsGsGenReq from FUI

Desired Response:

- Ground Script generated and stored with DMS
- FoGsStatus sent back to FUI

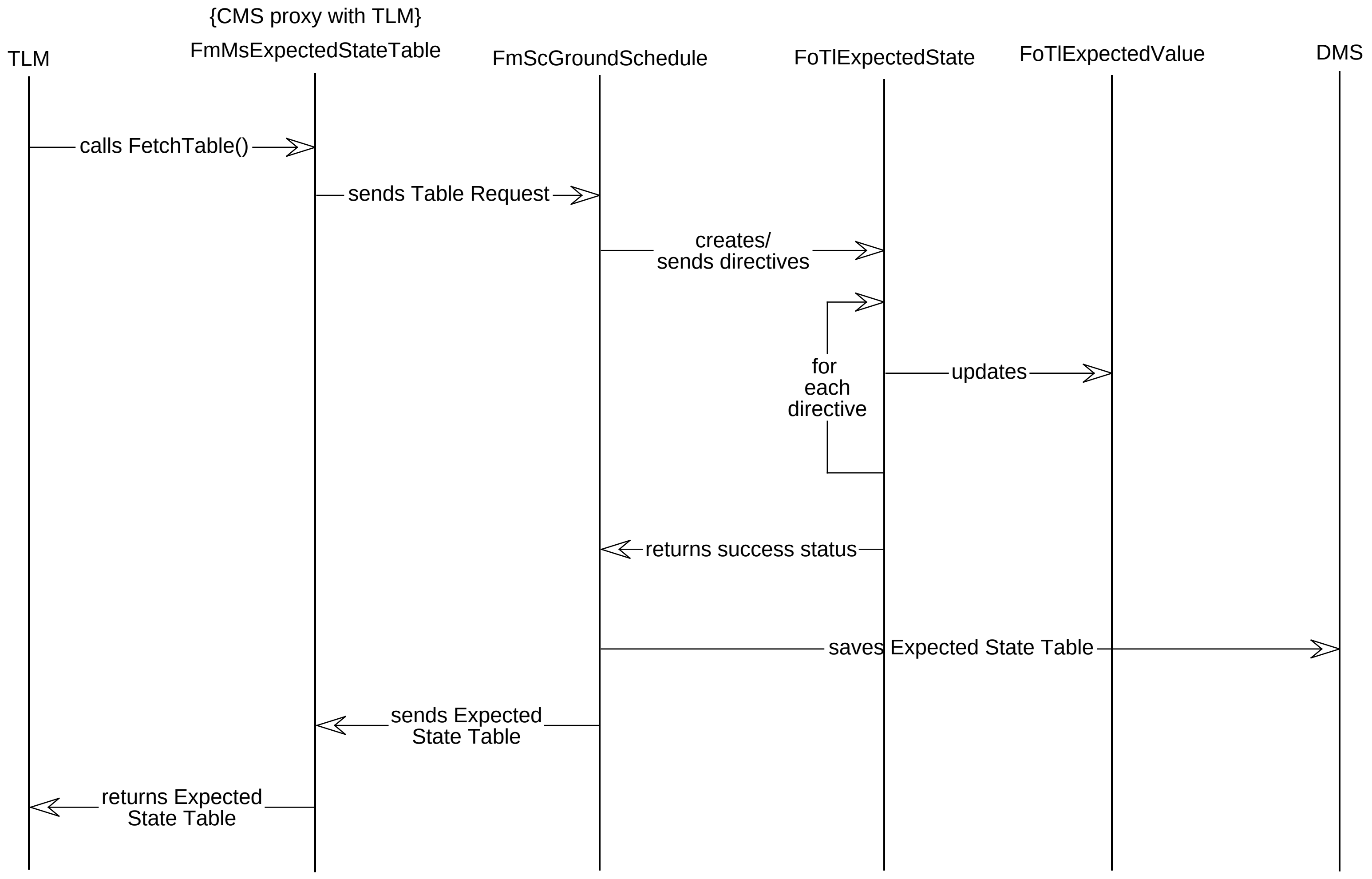


Figure 3.3-8. Expected State Table Generation Event Trace

Pre-Conditions:

- Ground Schedule software has been initialized
- Schedule has been populated with directives

Post-Conditions:

- none

3.3.4.5.3 Ground Script Generation Description

Figure 3.3-9 shows the Ground Script Generation Event Trace. The Ground Schedule receives an FoMsGsGenReq from the proxy FmMsGenerateOpAids. The Ground Schedule uses the information in the request to determine which directives from the schedule should be copied into a Ground Script. The Ground Script is created with these directives and stored with DMS. Ground Schedule creates an FoGsStatus object and indicates in it information pertaining to the generation. The FoGsStatus is returned to the proxy, which returns it to FUI.

3.3.5 Ground Schedule Data Dictionary

FmGsGroundData

`class` **`FmGsGroundData`**

Base Classes

`public` **`RWCollectable`**

Public Functions

`EcTVoid` **`DeleteDirectives`**`(const RWSlistCollectables&)`

Called by the Schedule Controller to delete certain directives from the Ground Schedule.

`EcTVoid` **`DeliverDirectives`**`(const FmMnDirectiveList&)`

Called by the Schedule Controller to put the directives from a DAS into the Ground Schedule.

`FmMnDirectiveList` **`ReturnCCList`**`(const RWTime&, const RWSListCollectables&)`

Called by the Schedule Controller to request a list of certain directives from the Ground Schedule.

Private Functions

`EcTInt` **`CreateConnection`**`()`

Creates a connection between this proxy and the Ground Schedule.

`EcTVoid` **`DestroyConnection`**`()`

Destroys the connection between this proxy and the Ground Schedule.

`RWCollectable` **`Receive`**`()`

Receives an object by ipc from the Ground Schedule.

`EcTVoid` **`Send`**`(const RWCollectable&)`

Ipc's an object to the Ground Schedule.

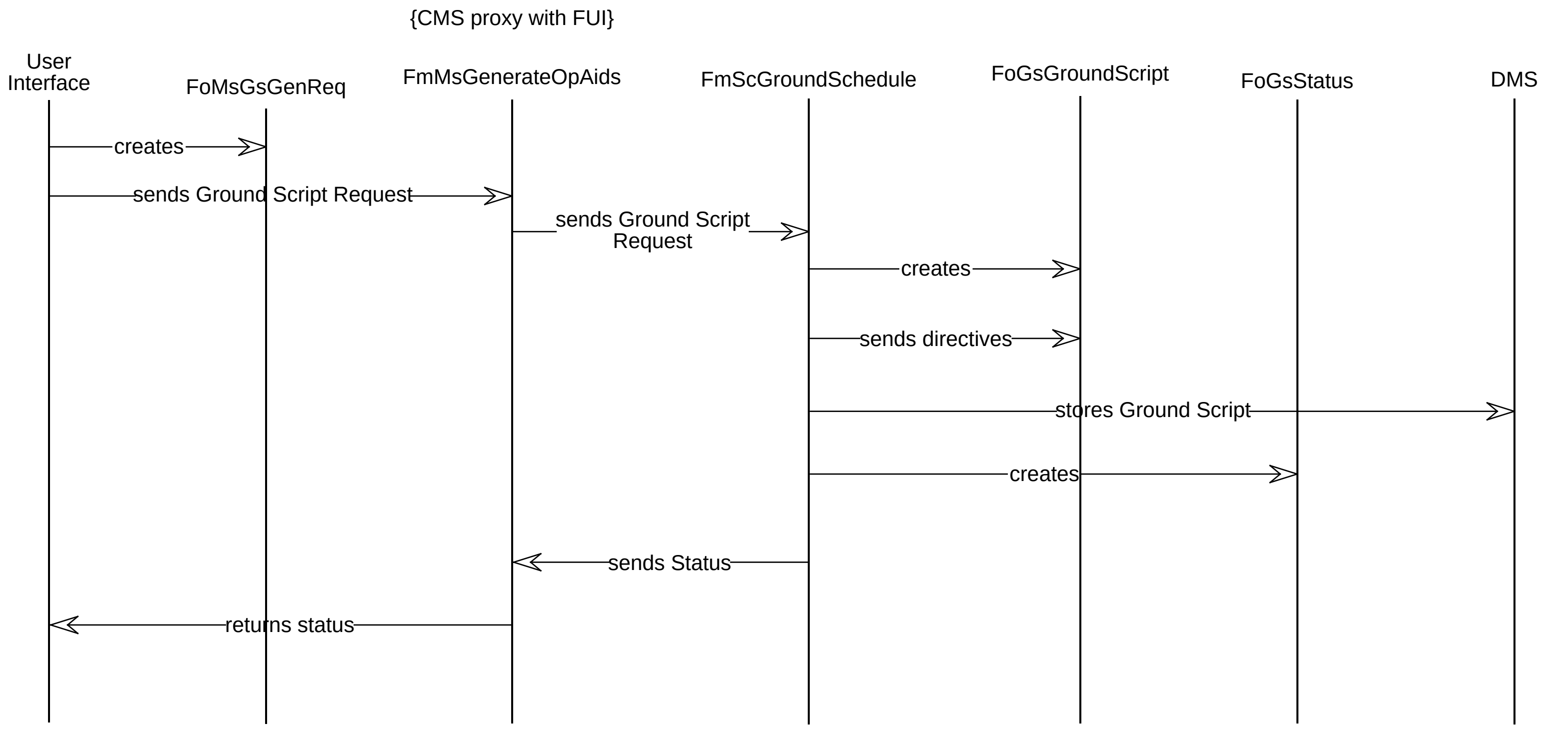


Figure 3.3-9. Ground Script Generation Event Trace

FmGsListRequest

class **FmGsListRequest**

This class represents the criteria for specifying which directives should be returned from the Ground Schedule. Only those directives having a DAS Id in myDASList and an execution time after myTime will be returned.

Base Classes

`public` **`RWCollectable`**

Private Data

`RWSlistCollectables` **`myDASList`**

A list of DAS id's.

`RWTime` **`myTime`**

The time for specifying which directives should be returned.

FmMsExpectedStateTable

class **FmMsExpectedStateTable**

Base Classes

`public` **`RWCollectable`**

Public Functions

`FoTlExpectedState&` **`FetchTable`**`()`

Called by TLM to request the generation of an expected state table.

Private Functions

`EcTInt` **`CreateConnection`**`()`

Establishes a connection between this proxy and the Ground Schedule. Returns TRUE is the connection was successful.

`EcTVoid` **`DestroyConnection`**`()`

Removes the connection between this proxy and the Ground Schedule.

`FoTlExpectedState` **`Receive`**`()`

Receives an Expected State Table from the Ground Schedule and returns it.

`EcTVoid` **`Send`**`(const FoMsTableRequest&)`

Sends a request to the Ground Schedule.

FmMsGenerateOpAids

class **FmMsGenerateOpAids**

This class represents an interface between the Ground Schedule and FUI. Its purpose is to provide FUI with a means to get information from the Ground Schedule. This class is able to send and receive certain objects via an ipc mechanism.

Public Functions

`FoGsStatus` **`GenerateGndScript`**`(const FoMsGsGenReq&)`

Send a generate ground script request object to the Ground Schedule and receives a status on the generation in return

Private Functions

`EcTInt` **`CreateConnection`**`()`

Establishes the connection between this proxy and the Ground Schedule

`EcTVoid` **`DestroyConnection`**`()`

Removes the connection between this proxy and the Ground Schedule

`FoGsStatus` **`Receive`**`()`

Receives the ground script status object from the Ground Schedule and returns it

`EcTVoid` **`Send`**`(const FoMsGsGenReq&)`

Sends the Ground Script Request object to the Ground Schedule

FmScGroundSchedule

`class` **`FmScGroundSchedule`**

This class represents the ground schedule. It is a time-ordered list of commands, representing seven operational days of the spacecraft. The class is responsible for maintaining the list of commands, generating reports, generating ground scripts, and creating and updating the expected state table.

Base Classes

`public` **`RWCollectable`**

Public Functions

`FoTlExpectedState&` **`CreateEST`**`(const RWTime&)`

Creates the expected state table object for a given time, adds it to myTableList, and writes it to a file in DMS.

`FoGsStatus` **`CreateGroundScript`**`(const FoMsGsGenRequest&)`

Makes a FoGsGroundScript object from the schedule upon request. Stores the script to a file.

`EcTVoid` **`CreateIntegratedReport`**`()`

Creates the integrated report. Whenever a new DAS is added to the schedule, a report is generated for the previous DAS.

`EcTVoid` **`DeleteDirectives`**`(const RWSlistCollectables&)`

Deletes all directives with the specified DASId from the schedule.

`EcTVoid` **`HandleMessage`**`()`

Handles all ipc messaging for the Ground Schedule.

`EcTVoid` **`Initialize`**`()`

Creates the empty schedule and table list. Populates the CEV Table.

`EcTInt` **`ProcessDAS`**`(const FmScDAS&)`

Adds the DAS to the schedule. Creates an expected state table and an integrated report.

`EcTVoid` **`ProcessOrbitalEvents`**`(const RWSlistCollectables&)`

Adds the orbital events to the ground schedule.

`EcTVoid` **`ProcessUplinkSched`**`(const FmScUplinkSched&)`

Adds the uplink commands to the ground schedule.

`FmMnDirectiveList&` **`ReturnCCList`**`(const FmGsListRequest&)`

Creates a special list of directives to use against a DAS for constraint checking. This list is given to the Schedule Controller.

Private Data

`FoGsCEVTable` **`myCEVTable`**

A table of Command Execution Verification records, used to create Expected State Tables.

`FoEvEvent*` **`myEventPtr`**

A pointer to an FoEvEvent object.

`FmMnDirectiveList` **`mySchedule`**

A doubly-linked list of FoEcDirectives.

`FoTlExpectedState` **`myExpectedState`**

A table representing the state of the spacecraft at a certain point in time.

FoGsCEVDataField

`class` **`FoGsCEVDataField`**

Base Classes

`public` **`RWCollectable`**

Private Data

`EcTInt` **`myCEVpid`**

A unique number identifying the telemetry parameter.

`RWCString` **`myCmdMnemonic`**

The mnemonic of the command associated with the CEVpid.

`EcTInt` **`myHighValue`**

The highest possible value allowed for this data field.

`EcTInt` **`myLowValue`**

The lowest possible value allowed for this data field.

FoGsCEVTable

`class` **`FoGsCEVTable`**

class definition - This class represents a table containing information about command execution verification values. The table consists of a number of data fields, each with the information about the CEV values.

Public Functions

`FoGsCEVDataField` **`LookUp`**`(const RWCString&)`

Look in the table for the given command mnemonic, and return the data field having that command mnemonic.

Private Data

`RWSlistCollectables` **`myCEVData`**

A list of FoGsCEVDataFields belonging with this table.

FoGsGroundScript

`class` **`FoGsGroundScript`**

This class represents the ground script.

`It is generated and stored as a file with DMS.`

Base Classes

`public` **`FoDsFile`**

Private Data

`FmMnDirectiveList` **`myDirectives`**

A list of directives for the Ground Script.

`RWCString` **`mySpacecraftId`**

The id of the spacecraft for which ground script is generated.

`RWTime` **`myStartTime`**

The starting time of the ground script.

`RWTime` **`myStopTime`**

The ending time of the ground script.

FoGsStatus

`class` **`FoGsStatus`**

Base Classes

`public` **`RWCollectable`**

Private Data

`RWCString` **`myDirectory`**

The name of the directory where the Ground Script was stored.

`RWCString` **`myFilename`**

The filename of the Ground Script.

`EcTInt` **`myStatus`**

Textual information about the generation of the Ground Script, including any errors that were encountered,

FoMsGsGenReq

`class` **`FoMsGsGenReq`**

Base Classes

`public` **`RWCollectable`**

Private Data

`RWCString` **`myDirectory`**

The name if the directory where the Ground Script should be stored.

`RWCString` **`myFilename`**

The filename that the Ground Script should be given.

`EcTInt` **`myProcExpFlag`**

An indicator of whether or not to include expanded procedures in the Ground Script.

`RWCString` **`myScId`**

The id of the spacecraft for the Ground Script.

`EcTLongInt` **`myStartTime`**

The time at which the Ground Script should start, implemented as a number of seconds.

`EcTLongInt` **`myStopTime`**

The time at which the Ground Script should end. If no end time is specified, the Ground Script will be generated up to the last command in the Schedule.

FoMsTableRequest

`class` **`FoMsTableRequest`**

Base Classes

`public` **`RWCollectable`**

Private Data

`RWTime` **`myTime`**

The time at which the Expected State Table should be generated. The default is now.

FoRpIntegratedReport

`class` **`FoRpIntegratedReport`**

This class represents the integrated report.

`It is generated from information in the Ground Schedule and`

stored as a file with DMS.

Base Classes

`public` **`FoDsFile`**

Private Data

`FmMnDirectiveList` **`myDirectiveList`**

A list of directives for the Integrated Report.

FoTlExpectedState

`class` **`FoTlExpectedState`**

class definition

Base Classes

`public` **`RWCollectable`**

Public Functions

`EcTVoid` **`Compare`**`(const FoPsClientBuffer&)`

Compares the input table to this table, and puts the result of the comparison in an event message.

`RWSlistCollectables` **`GetPids`**`()`

Returns a list of pids that exist in this table.

`EcTInt` **`Replace`**`(const FoPsClientBuffer&)`

Replaces the values in this table with the values in the input table.

`EcTInt` **`UpdateTable`**`(const RWDlistCollectables&)`

Updates the table using the CEV table and the input directives.

Private Data

`FoGsCEVTable*` **`myCEVTable`**

A pointer to the CEV table, gotten from the database by the Ground Schedule. The CEV table is used to generate a new Expected State Table.

`RWSlistCollectables&` **`myData`**

A list of data fields which hold all the information for for this table.

`RWTime` **`myTime`**

The time at which this table was generated.

FoTlExpectedValue

`class` **`FoTlExpectedValue`**

class definition

Base Classes

`public` **`RWCollectable`**

Private Data

`EcTInt` **`myHighValue`**

The highest possible value acceptable for this field. It can be either an integer or a float.

`EcTInt` **`myLowValue`**

The lowest possible value accepted for this field. It can be either an integer or a float.

`EcTInt` **`myPID`**

A unique number identifying the parameter id for this field.

3.4 Command Model

The Command Model component consists of two processes that run on the FOS Data Server: Command Model and Rule Constraint Check. Together these processes are responsible for performing rule-based constraint checking on lists of commands.

The Command Model process is persistent and is responsible for handling interprocess communication with other subsystems and with other CMS processes. Command Model receives requests for constraint checking of activity definitions, RTS definitions, command procedure definitions, and command lists representing scheduled activities. After the constraint check is complete, Command Model returns a status, including a list of constraint violations, to the requesting process.

Rule Constraint Check is a temporary process spawned by Command Model to actually perform the constraint checking. Constraint definitions are associated in the FOS database with the commands that they apply to. Rule Constraint Check examines the list of commands passed to it and, for each command having an associated constraint, performs the constraint check.

3.4.1 Command Model Context

The CMS Command Model interfaces with several FOS subsystems and the CMS Schedule Controller task, as shown in the Context Diagram (see Figure 3.4-1) and summarized below.

CMS Schedule Controller:

- Sends a Directive List, which contains a sequence of ATC, RTS, and real-time commands.
- Receives a Constraint Check Status, which includes information on any directive that violated a database defined constraint.

CMS Load Catalog:

- Sends an RTS Load Contents, which contains a sequence of commands with relative time tags.
- Receives a Constraint Check Status, which includes information on any directive in an RTS load contents that violated a database defined constraint.

User Interface:

- Sends a Command Procedure, which contains a sequence of ECL directives including real-time commands.
- Receives a Constraint Check Status, which includes information on any directive in a procedure that violated a database defined constraint.

Data Management:

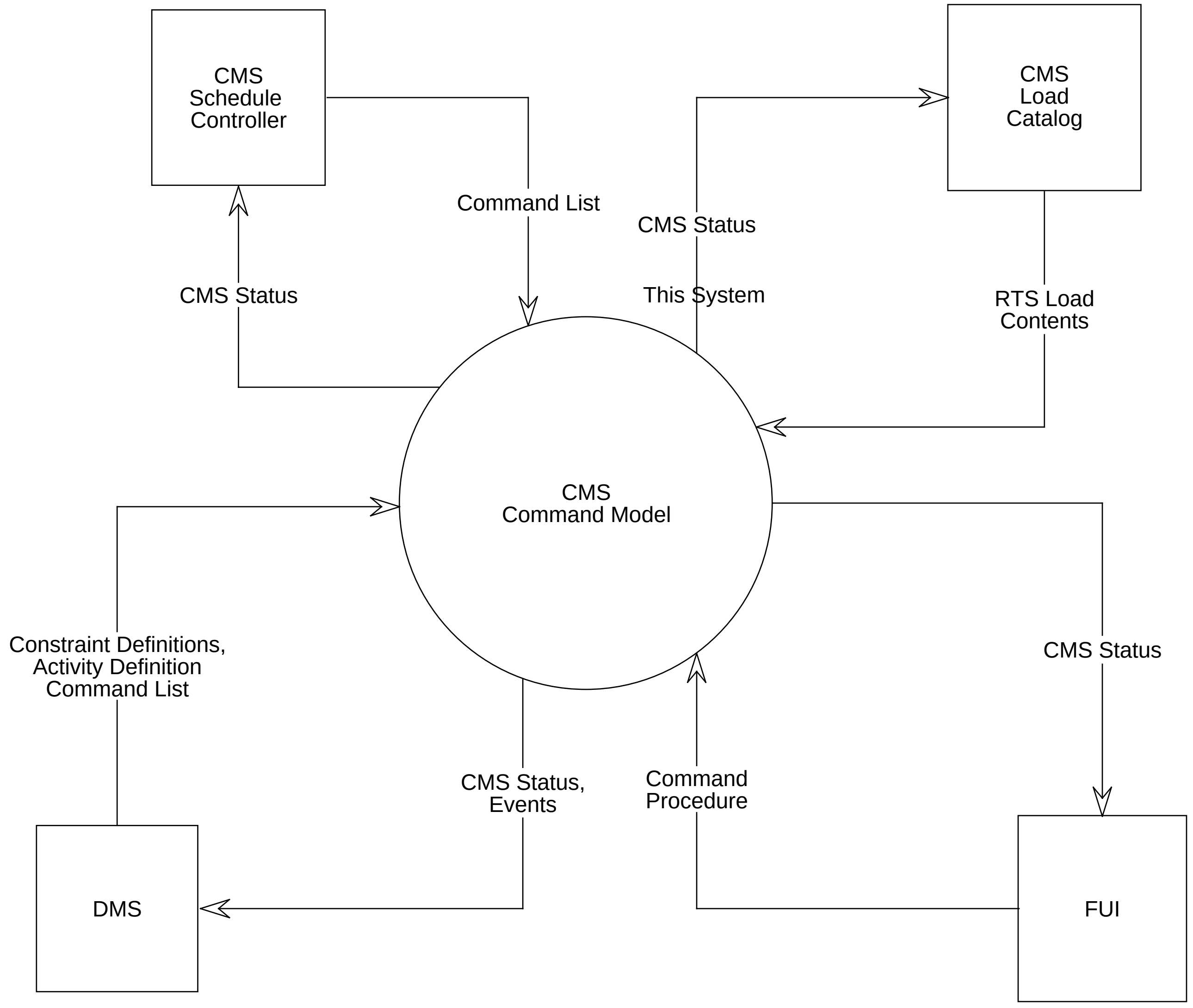


Figure 3.4-1. Command Model Context Diagram

- Sends an Activity Definition Directive List, which contains a sequence of commands with relative time tags.
- Receives a Constraint Check Status, which includes information on any directive in an activity definition that violated a database defined constraint.
- Provides Constraint Definitions, which specify the rules being checked.
- Receives Events, which are status messages about CMS Command Model processing.

3.4.2 Command Model Interfaces

Table 3.4.2. Command Model Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Validate Activities	FmMsValidateActivities	Proxy between Command Model and DMS. Receives a list of activity definitions to be constraint checked. Result list of FoMsCMSStatus is returned	CMS: Command Model	DMS: PDB Validation	1/week
Validate Procedure	FmMsValidateProcedure	Proxy between Command Model and FUI	CMS: Command Model	FUI: Procedure Builder	1/week
	FoMSValidateProcReq	Request to validate given procedure			
	FoMsCMSStatus	Status of constraint check			
Validate DAS Command List	FmMsValidateConstraints	Proxy between CMS: Command Model and CMS: Schedule Controller	CMS: Command Model	CMS:Schedule Controller	1/day
	FmScConstCk	Command List to be constraint checked			
	FoMsCMSStatus	Status of constraint check			
Validate RTS load	FmMsValidateConstraints	Proxy between CMS: Command Model and CMS: Load Catalog. Proxy retrieves RTS load to be constraint checked	CMS: Command Model	CMS: Load Catalog	1/day
	FoMsCMSStatus	Status of constraint check			
Send Events	FoFdEventLogger	Proxy between DMS and CMS: Spacecraft requesting broadcast of a message	DMS: Event Logger	CMS: Spacecraft	10/day

3.4.3 Command Model Object Model

FmCcCommandModel controls the processing of requests for rule-based command-level constraint checking of scheduled commands, procedure definitions, RTS load definitions, and activity definitions (see figures 3.4-2 through 3.4-8). FmCcCommandModel passes the FmScConstCk list of scheduled commands received from the CMS Schedule Controller directly to FmCcRuleConstraintModel for constraint checking. For procedure, RTS, or activity definitions, FmCcCommandModel creates an FmCcDirectiveList from the definition and passes it to FmCcRuleConstraintModel.

To process a request for checking an RTS definition, FmCcCommandModel expands the RTS definitions and forwards the directive list to be constraint checked.

To process a request for checking a procedure definition, FmCcCommandModel makes a single FmCcDirectiveList for each possible path in a procedure. The directive lists include the further expansion of procedures called from within the definition being validated. Each of these lists are sent to FmCcRuleConstraintModel for command rule-based constraint checking. The list of FoMsConflictInfos received in FoMsCMSStatus for each list are combined to make a single FoMsCMSStatus. The worst status returned will determine the newly created FoMsCMSStatus that is forwarded to FmMsValidateProcedure proxy.

To process a request for checking an activity definition, FmCcCommandModel processes of list of activity definitions. Each activity definition is expanded into a FmCcDirectiveList. Procedures within activities are also expanded as described above, this will create multiple FmCcDirectiveLists per activity definition. Each list is command rule-based constraint checked. The FoMsConflictInfo for each directive list is combined so that there is one FoMsCMSStatus for each activity definition. A list of FoMsCMSStatuses is returned to FmMsValidateActivities proxy.

The proxies, FmMsValidateConstraints, FmMsValidateProcedure, and FmMsValidateActivites, provide interprocess communication (IPC) between the FmCcCommandModel and its various interfaces.

FmMsValidateConstraints manages communications with the CMS Schedule Controller (FmScScheduleController) and the CMS Load Catalog (FmLdLoadCatalog). FmMsValidateConstraints receives a FmScConstCk command list from the schedule controller which is forwarded to FmCcCommandModel. It also receives the RTS file name and location. From this information it creates a FmCcDirectiveList from the RTS directives and this list is sent to FmCcCommandModel for processing.

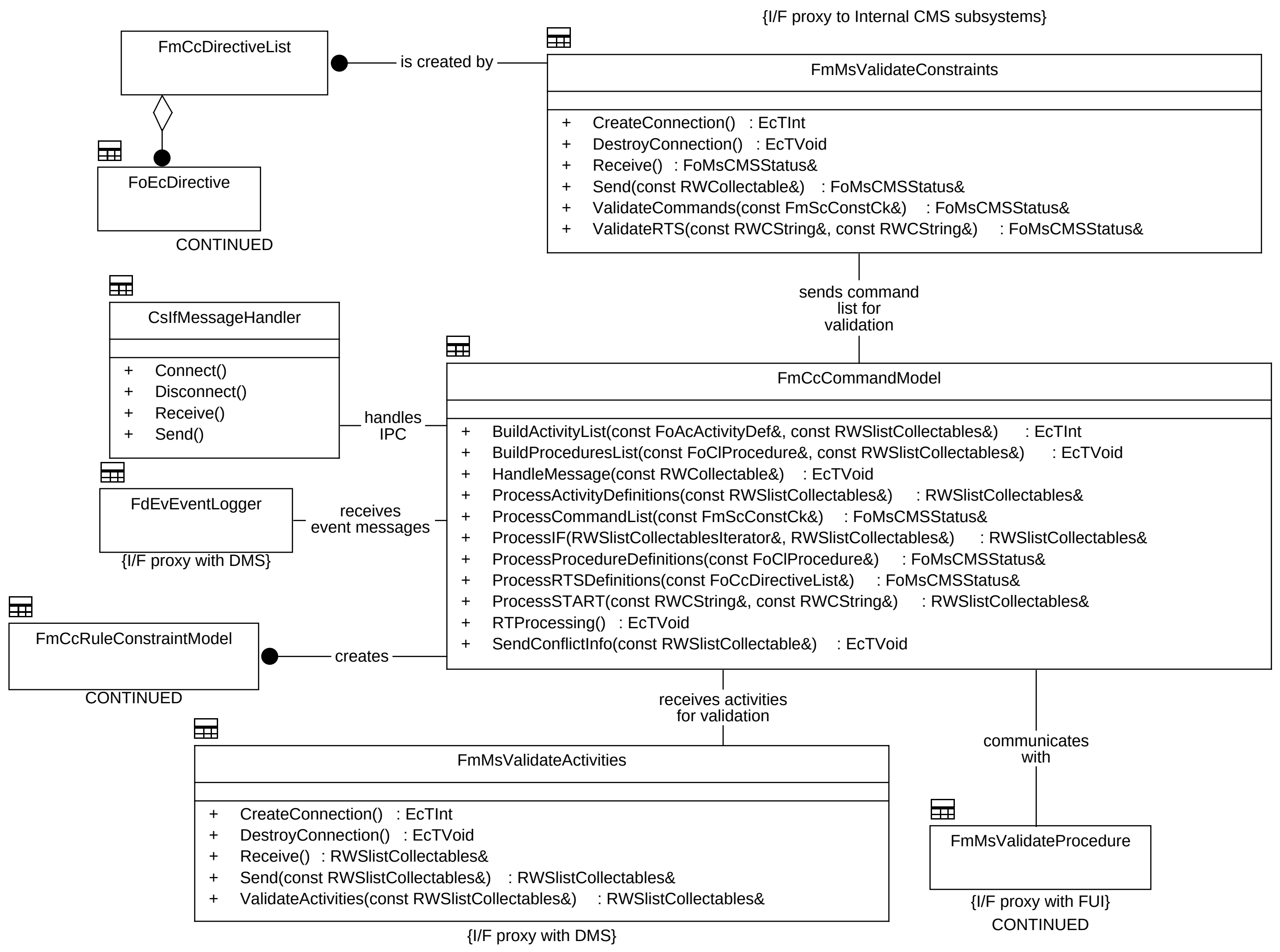

{I/F proxy to Internal CMS subsystems}
FmCcDirectiveList
is created by
FmMsValidateConstraints
+ CreateConnection() : EcTInt
+ DestroyConnection() : EcTVoid
+ Receive() : FoMsCMSStatus&
+ Send(const RWCollectable&) : FoMsCMSStatus&
+ ValidateCommands(const FmScConstCk&) : FoMsCMSStatus&
+ ValidateRTS(const RWCString&, const RWCString&) : FoMsCMSStatus&
FoEcDirective
CONTINUED
sends command list for validation
CsIfMessageHandler
+ Connect()
+ Disconnect()
+ Receive()
+ Send()
handles IPC
FmCcCommandModel
+ BuildActivityList(const FoAcActivityDef&, const RWSlistCollectables&) : EcTInt
+ BuildProceduresList(const FoClProcedure&, const RWSlistCollectables&) : EcTVoid
+ HandleMessage(const RWCollectable&) : EcTVoid
+ ProcessActivityDefinitions(const RWSlistCollectables&) : RWSlistCollectables&
+ ProcessCommandList(const FmScConstCk&) : FoMsCMSStatus&
+ ProcessIF(RWSlistCollectablesIterator&, RWSlistCollectables&) : RWSlistCollectables&
+ ProcessProcedureDefinitions(const FoClProcedure&) : FoMsCMSStatus&
+ ProcessRTSDefinitions(const FoCcDirectiveList&) : FoMsCMSStatus&
+ ProcessSTART(const RWCString&, const RWCString&) : RWSlistCollectables&
+ RTProcessing() : EcTVoid
+ SendConflictInfo(const RWSlistCollectable&) : EcTVoid
FdEvEventLogger
receives event messages
{I/F proxy with DMS}
FmCcRuleConstraintModel
creates
CONTINUED
receives activities for validation
communicates with
FmMsValidateActivities
+ CreateConnection() : EcTInt
+ DestroyConnection() : EcTVoid
+ Receive() : RWSlistCollectables&
+ Send(const RWSlistCollectables&) : RWSlistCollectables&
+ ValidateActivities(const RWSlistCollectables&) : RWSlistCollectables&
{I/F proxy with DMS}
FmMsValidateProcedure
{I/F proxy with FUI}
CONTINUED

Figure 3.4-2. Command Model Object Model (1 of 7)

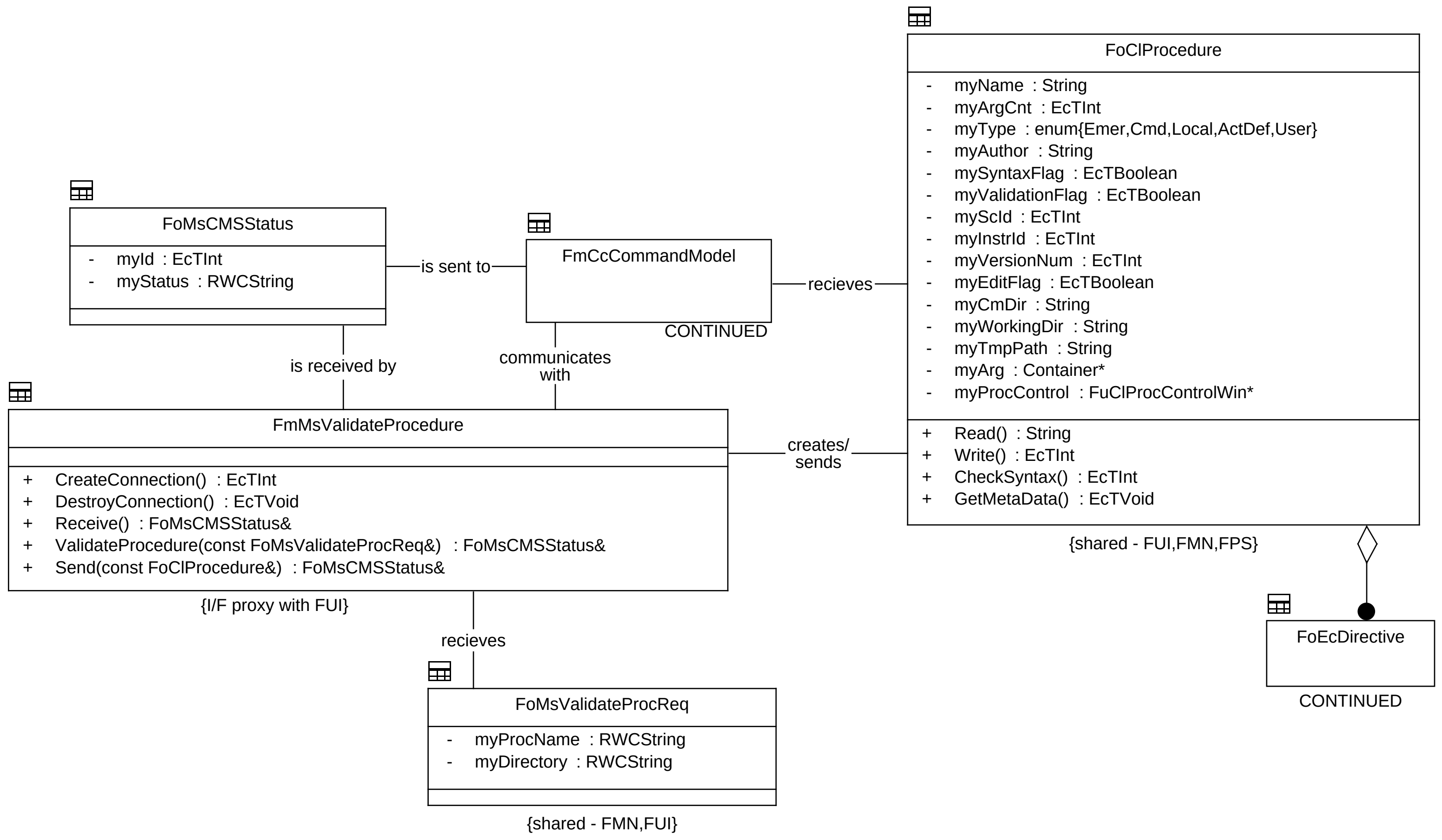


Figure 3.4-3. Command Model Object Model (2 of 7)

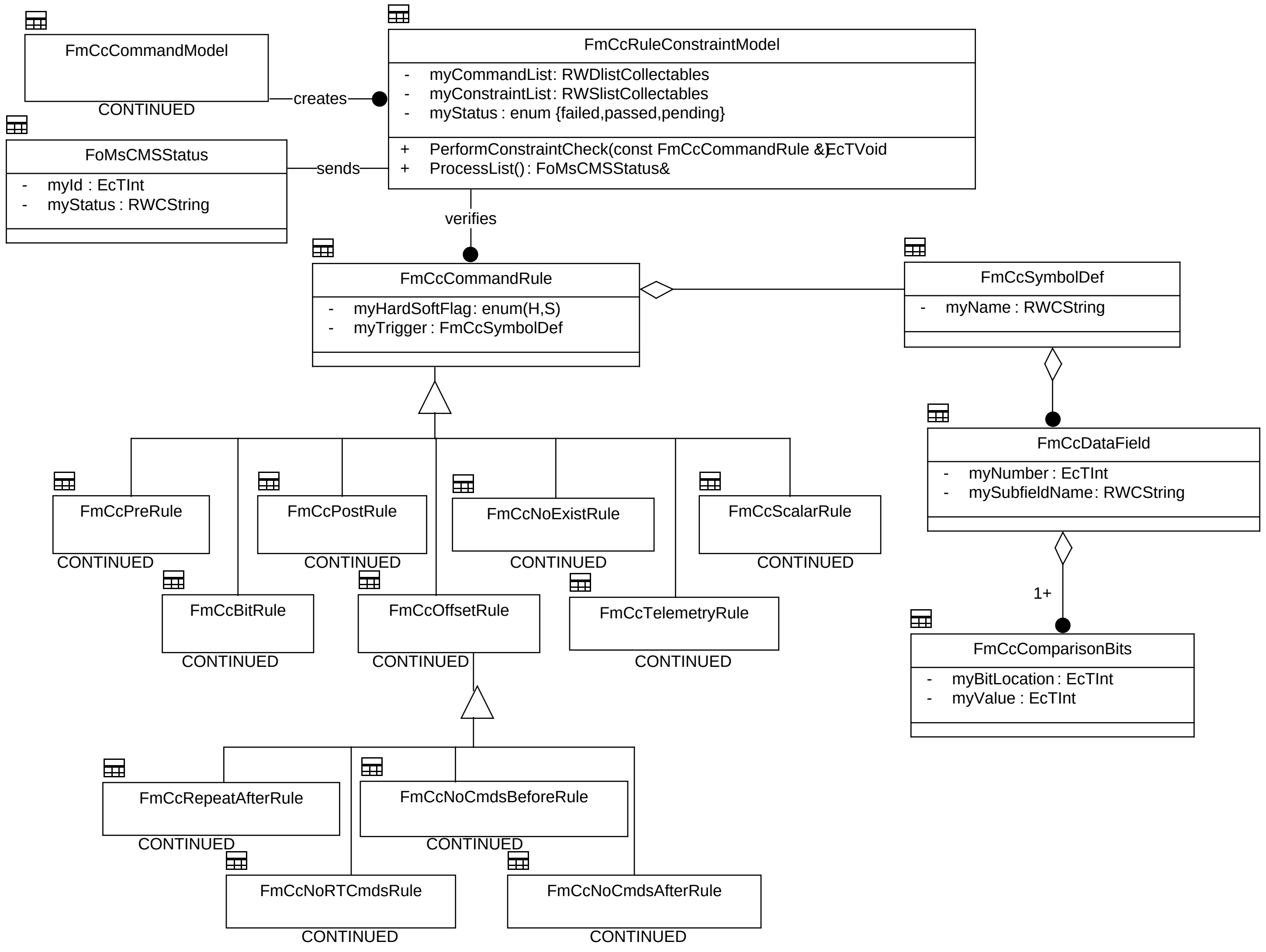
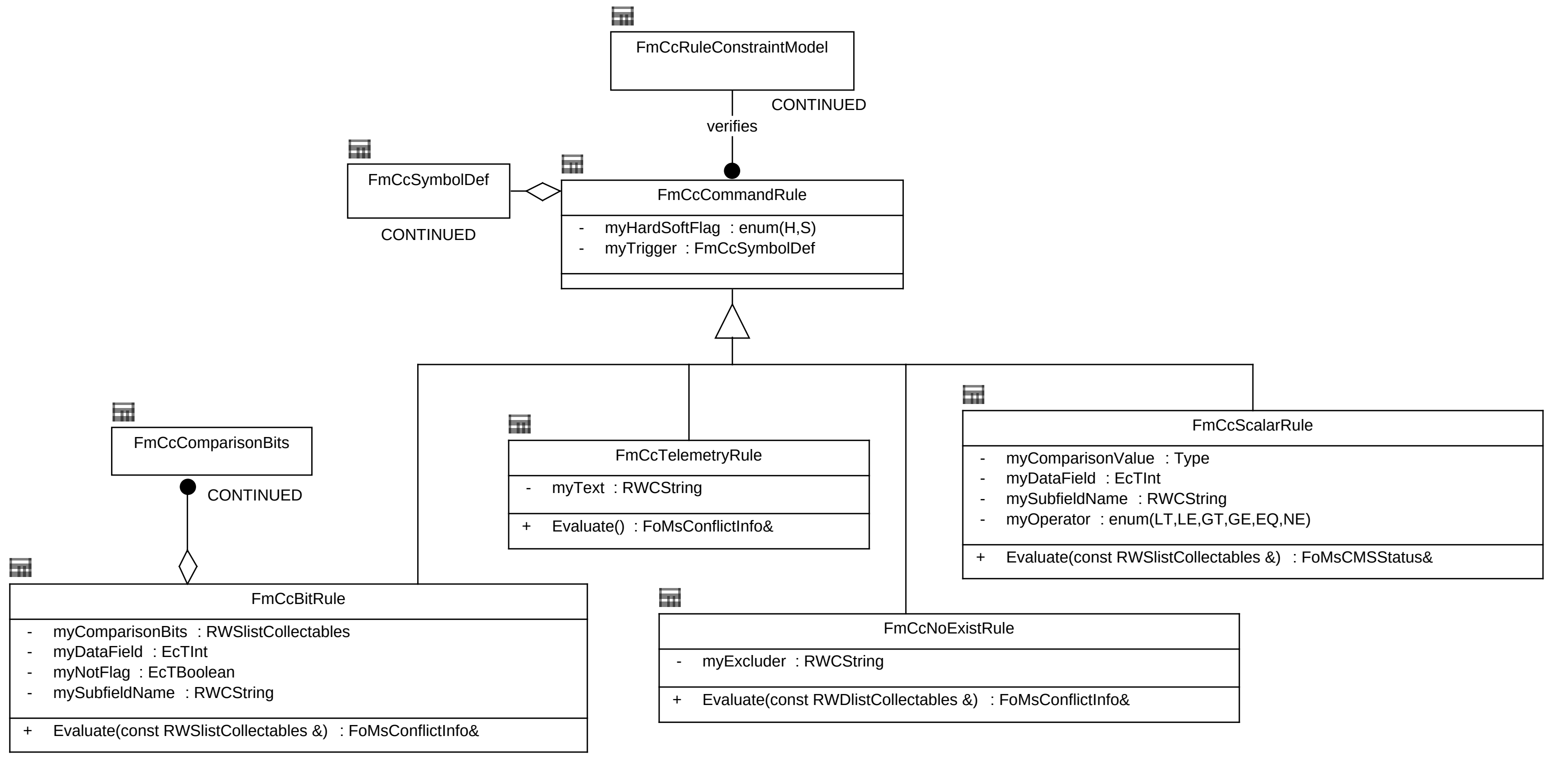


Figure 3.4-4. Command Model Object Model (3 of 7)



FmCcRuleConstraintModel
CONTINUED
verifies
FmCcSymbolDef
CONTINUED
FmCcCommandRule
- myHardSoftFlag : enum(H,S)
- myTrigger : FmCcSymbolDef
FmCcComparisonBits
CONTINUED
FmCcTelemetryRule
- myText : RWCString
+ Evaluate() : FoMsConflictInfo&
FmCcScalarRule
- myComparisonValue : Type
- myDataField : EcTInt
- mySubfieldName : RWCString
- myOperator : enum(LT,LE,GT,GE,EQ,NE)
+ Evaluate(const RWSlistCollectables &) : FoMsCMSStatus&
FmCcBitRule
- myComparisonBits : RWSlistCollectables
- myDataField : EcTInt
- myNotFlag : EcTBoolean
- mySubfieldName : RWCString
+ Evaluate(const RWSlistCollectables &) : FoMsConflictInfo&
FmCcNoExistRule
- myExcluder : RWCString
+ Evaluate(const RWDlistCollectables &) : FoMsConflictInfo&

Figure 3.4-5. Command Model Object Model (4 of 7)

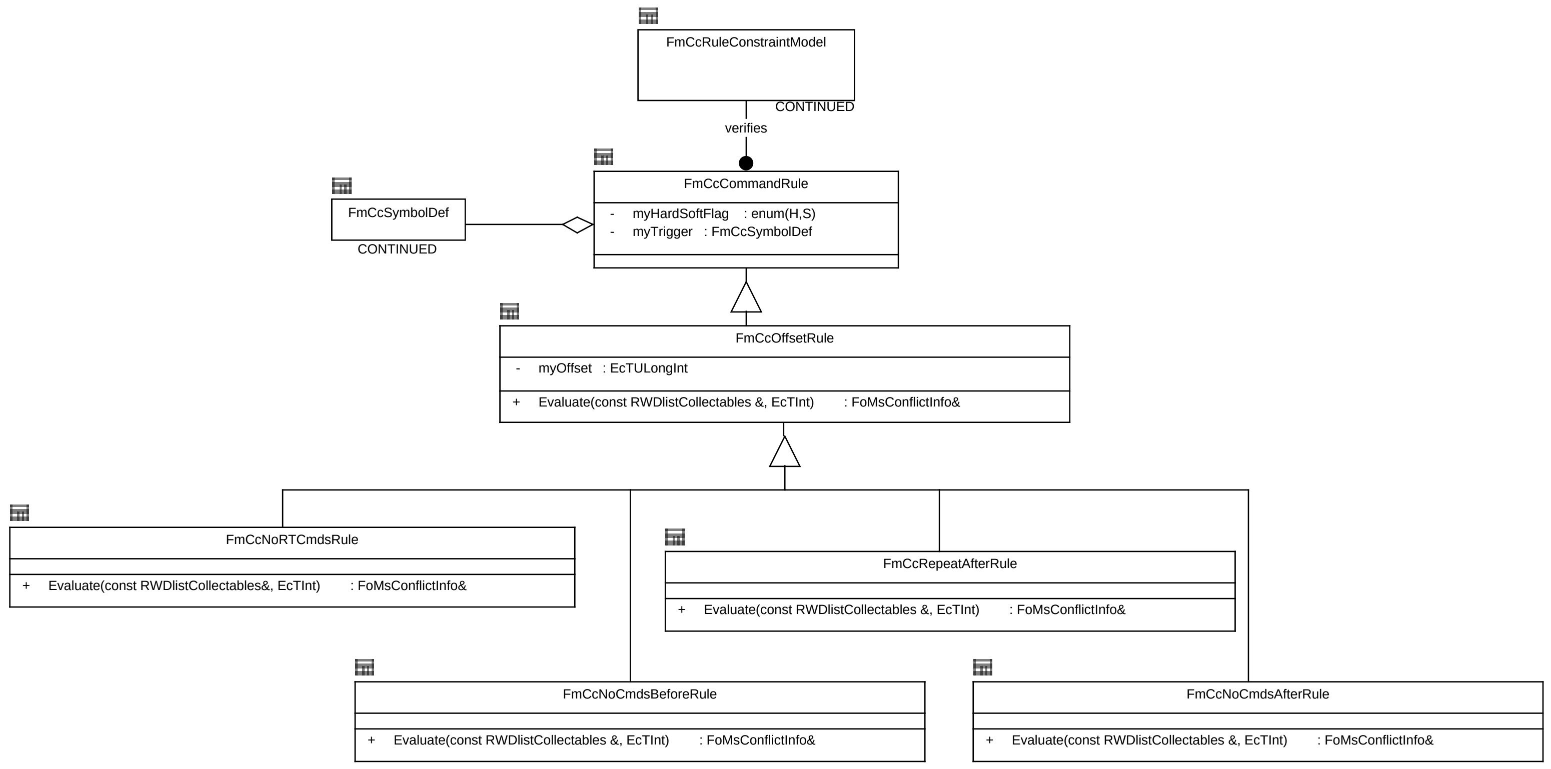

FmCcRuleConstraintModel
CONTINUED
verifies
FmCcSymbolDef
CONTINUED
FmCcCommandRule
- myHardSoftFlag : enum(H,S)
- myTrigger : FmCcSymbolDef
FmCcOffsetRule
- myOffset : EcTULongInt
+ Evaluate(const RWDlistCollectables &, EcTInt) : FoMsConflictInfo&
FmCcNoRTCmdsRule
+ Evaluate(const RWDlistCollectables&, EcTInt) : FoMsConflictInfo&
FmCcRepeatAfterRule
+ Evaluate(const RWDlistCollectables &, EcTInt) : FoMsConflictInfo&
FmCcNoCmdsBeforeRule
+ Evaluate(const RWDlistCollectables &, EcTInt) : FoMsConflictInfo&
FmCcNoCmdsAfterRule
+ Evaluate(const RWDlistCollectables &, EcTInt) : FoMsConflictInfo&

Figure 3.4-6. Command Model Object Model (5 of 7)

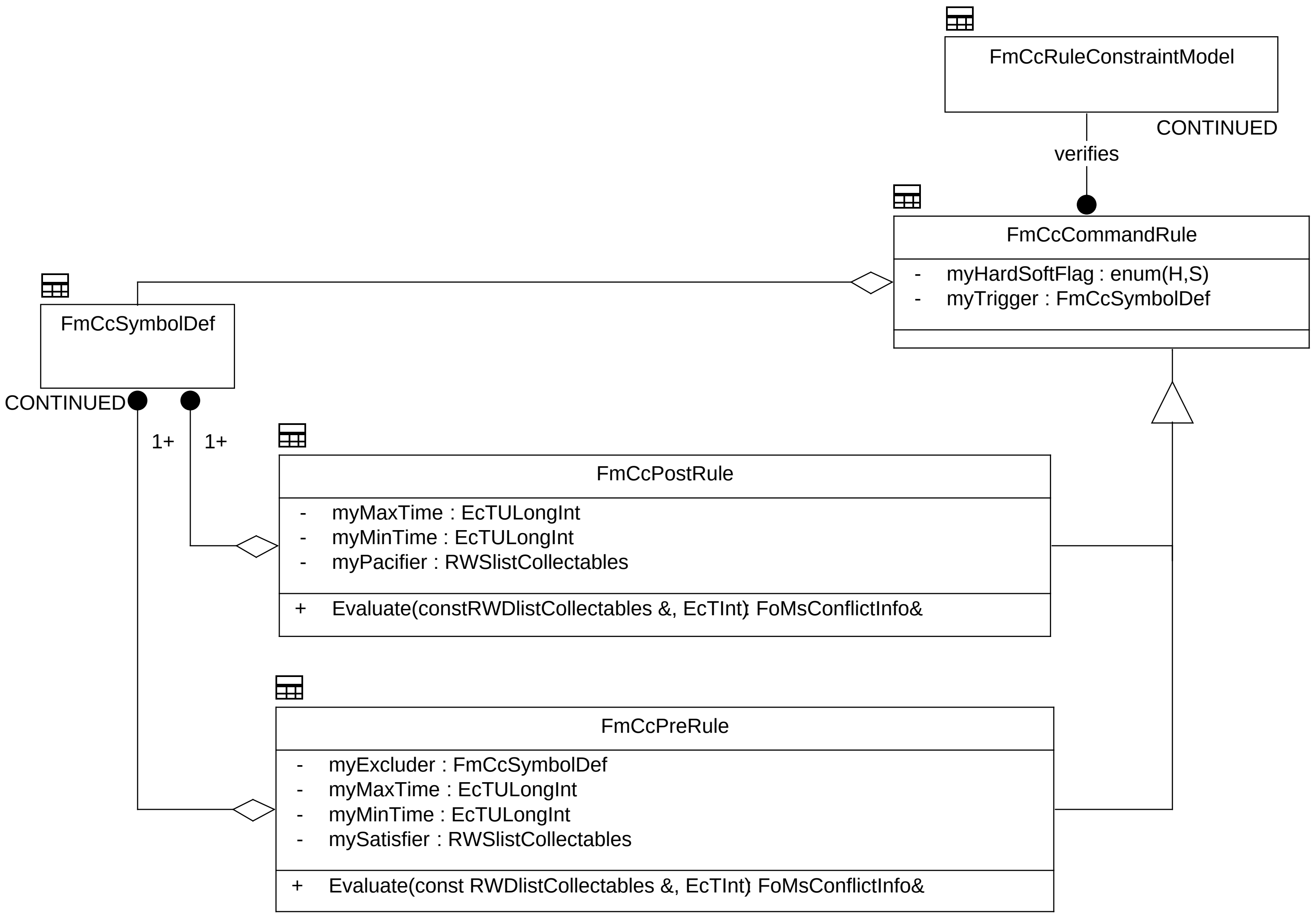

FmCcRuleConstraintModel
CONTINUED
verifies
FmCcCommandRule
- myHardSoftFlag : enum(H,S)
- myTrigger : FmCcSymbolDef
FmCcSymbolDef
CONTINUED
1+
1+
FmCcPostRule
- myMaxTime : EcTULongInt
- myMinTime : EcTULongInt
- myPacifier : RWSlistCollectables
+ Evaluate(constRWDlistCollectables &, EcTInt) : FoMsConflictInfo&
FmCcPreRule
- myExcluder : FmCcSymbolDef
- myMaxTime : EcTULongInt
- myMinTime : EcTULongInt
- mySatisfier : RWSlistCollectables
+ Evaluate(const RWDlistCollectables &, EcTInt) : FoMsConflictInfo&

Figure 3.4-7. Command Model Object Model (6 of 7)

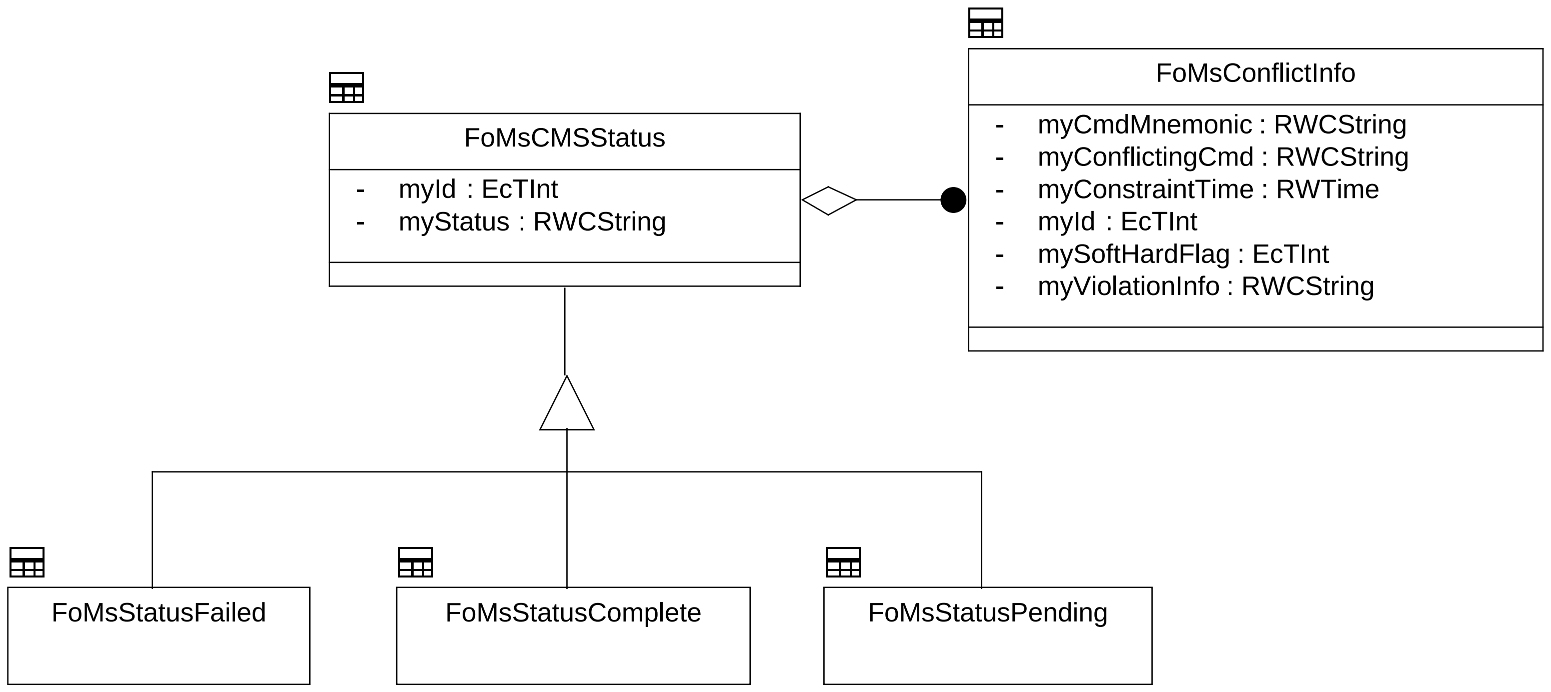

FoMsCMSStatus
- myId : EcTInt
- myStatus : RWCString
FoMsConflictInfo
- myCmdMnemonic : RWCString
- myConflictingCmd : RWCString
- myConstraintTime : RWTime
- myId : EcTInt
- mySoftHardFlag : EcTInt
- myViolationInfo : RWCString
FoMsStatusFailed
FoMsStatusComplete
FoMsStatusPending

Figure 3.4-8. Command Model Object Model (7 of 7)

FmMsValidateProcedure manages communication with FUI. FmMsValidateProcedure receives the location of the procedure definition and instantiates the FoClProcedure class which is sent to FmCcCommandModel for processing.

FmMsValidateActivities manages communication with DMS. FmMsValidateActivities receives a list of activity definitions. This list is forwarded to FmCcCommandModel for processing.

FmCcRuleConstraintModel processes the directive lists. It keeps track of the number of ATC commands issued in the same second and verifies the existence of a load when an uplink request is detected. If a directive in the list has a constraint associated with it , FmCcRuleConstraintModel performs the validation of that database defined command constraint rule. Depending on the requirement of the rule, the directive list can be searched forward or backward to locate (or not) the command that will satisfy the rule. FmCcRuleConstraintModel returns an FoMsCMSStatus.

FmCcSymbolDef is associated with the different FmCcCommandRules. It allows certain triggers, excluders, satisfiers and pacifiers to be defined in greater detail. Each rule has a symbol definition. If a FmCcSymbolDef doesn't have FmCcDataFields defined for it, then the trigger, excluded, satisfier, or pacifier is the command itself, i.e. the command need only exist and not have its data fields set to a certain value. If there are multiple FmCcSymbolDefs defined (this is only allowed for satisfiers and pacifiers) then the rule may be validated if any one of the FmCcSymbolDef commands is found in the command list. If FmCcDataFields are defined then the bits for those data fields must match the command in the list in order to validate the constraint on the current command (trigger).

Triggers (the current command being constraint checked) for FmCcTelemetryRule, FmCcPreRule and FmCcPostRule may have FmCcDataFields defined, but may not have multiple instances of FmCcSymbolDef. All other FmCcCommandRule types may only have a single instance of FmCcSymbolDef without FmCcDataFields defined - the simplest form. The Excluded command in FmCcPreRule may have FmCcDataFields defined but may not have multiple FmCcSymbolDefs. The Excluded command in FmCcNoExistRule may only be defined in the simplest form of FmCcSymbolDef. Pacifiers and Satisfiers can have any one of the three instances of a FmCcSymbolDef.

FmCcCommandRule is an abstract class. All the rule types are derived from this class. FmCcCommandRule returns an FoMsConflictInfo when a violation occurs or NULL if the command rule is verified. Only one FmCcCommandRule is evaluated at any one time.

FmCcBitRule is derived from FmCcCommandRule. It verifies the rule specified data field FmCcComparisonBits are (or are not) equal to the data field of the trigger command.

FmCcNoCmdsAfterRule is derived from FmCcOffsetRule. It verifies that no commands are issued between the trigger command and the rule specified offset.

FmCcNoCmdsBeforeRule is derived from FmCcOffsetRule. It verifies that no commands are issued between the rule specified offset and the trigger command.

FmCcNoExistRule is derived from FmCcCommandRule. It verifies that the rule specified Excluded command is not in the command list.

FmCcNoRTCmdsRule is derived from FmCcOffsetRule. It verifies that after the trigger command is planned to be issued to a certain remote terminal (RT) that no other commands are issued to the same RT until the offset.

FmCcOffsetRule is derived from FmCcCommandRule. This is an abstract class. Any rule that has an offset is derived from this class.

FmCcPostRule is derived from FmCcCommandRule. It verifies that the given pacifier succeeds the trigger command in the given time period. The trigger may only have a single FmCcSymbolDef defined with optional FmCcDataFields specified. The pacifier may be defined using multiple FmCcSymbolDefs indicating that any one command in the list of FmCcSymbolDefs may satisfy the rule.

FmCcPreRule is derived from FmCcCommandRule. It verifies that the given satisfier proceeds the trigger command in the given time period. An optional excluded command may be defined and represents a command that cannot occur between the current command and the given satisfier. The trigger and the excluded may only have a single FmCcSymbolDef defined with optional FmCcDataFields specified. The satisfier may be defined using multiple FmCcSymbolDefs indicating that any one command in the list of FmCcSymbolDefs may satisfy the rule.

FmCcScalarRule is derived from FmCcCommandRule. It verifies that the trigger command data fields value properly compares to the FmCcDataField value stated in the rule. The following comparisons are permitted: less than, less than or equal to, greater than, greater than or equal to, equal to and not equal to.

FmCcTelemetryRule is derived from FmCcCommandRule. An FOT specified test string is output to FoMsConflictInfo. the specified trigger may have only one FmCcSymbolDef defined with optional FmCcDataFields.

FoMsCMSStatus is an abstract class that represents the results of command level rule-based constraint checking. It contains a list of FoMsConflictInfo objects.

FoMsConflictInfo specifies the information concerning the violation of a command level rule-based constraint check

FoMsStatusFailed is derived from FoMsCMSStatus. If any of the FoMsConflictInfo objects indicate a hard constraint this object is returned.

FoMsStatusPending is derived from FoMsCMSStatus. If all of the FoMsConflictInfo objects indicate soft constraints this object is returned.

FoMsStatusComplete is derived from FoMsCMSStatus. No FoMsConflictInfo objects were returned as a result of evaluating the rules indicating the directive list is valid.

3.4.4 Command Model Dynamic Model

The Command Model Dynamic Model has five scenarios:

- Initialization
- Expanded Directive List Receipt
- Command Procedure Receipt
- Activity Definition List Receipt
- RTS Load Definition Receipt

3.4.4.1 Command Model Initialization Scenario

3.4.4.1.1 Command Model Initialization Scenario Abstract

This scenario occurs when the Command Model process is started. It addresses the initialization of the Command Model interfaces and loading of configuration files (see Figure 3.4-9).

3.4.4.1.2 Command Model Initialization Summary Information

Interfaces:

- Name Server
- FUI
- DMS Event Handling
- DMS Sybase
- CMS Schedule Controller
- CMS Load Catalog

Stimulus:

- Command Model process is started

Desired Response:

- Command Model is ready to receive request to perform rule based command level constraint checking

Pre-Conditions:

- None

Post-Conditions:

•Command Model is ready to process requests

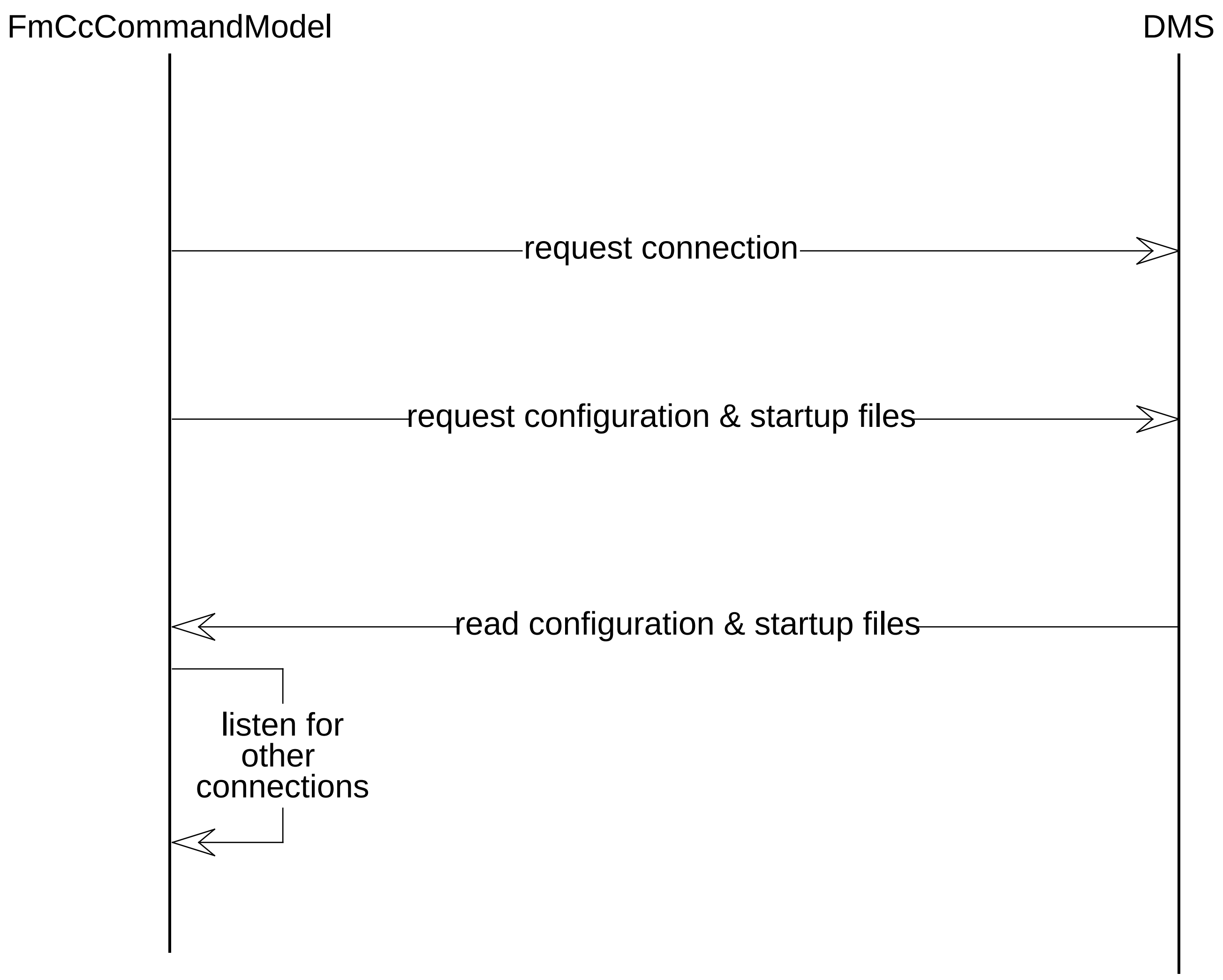

FmCcCommandModel
DMS
request connection
request configuration & startup files
read configuration & startup files
listen for other connections

Figure 3.4-9. Command Model Initialization Event Trace

3.4.4.1.3 Command Model Initialization Scenario Description

The Expanded Directive List Receipt scenario describes the receipt and processing of an When the Command Model is started FmCcCommandModel will initialize its interfaces by requesting address information from the Name Server. Once the interface connections have been made FmCcCommandModel will listen for requests.

3.4.4.2 Command Model Expanded Directive List Receipt Scenario

3.4.4.2.1 Command Model Expanded Directive List Receipt Scenario Abstract

Expanded Directive List from CMS Schedule (FmScScheduleController) via FmMsValidateConstraints proxy (see Figure 3.4-10).

3.4.4.2.2 Command Model Expanded Directive List Receipt Summary Information

Interfaces:

- CMS Schedule Controller

Stimulus:

- Receipt of Expanded Directive List

Desired Response:

- Status returned

Pre-Conditions:

- Schedule controller software has been initiate
- Command model software has been initiated

Post-Conditions:

- None

3.4.4.2.3 Command Model Expanded Directive List Receipt Scenario Description

The Command Model receives the expanded directive list from the Schedule Controller. It sends the directive list to the Rule Constraint Model for command rule-based constraint validation. An FoMsCMSStatus is returned to the Schedule Controller.

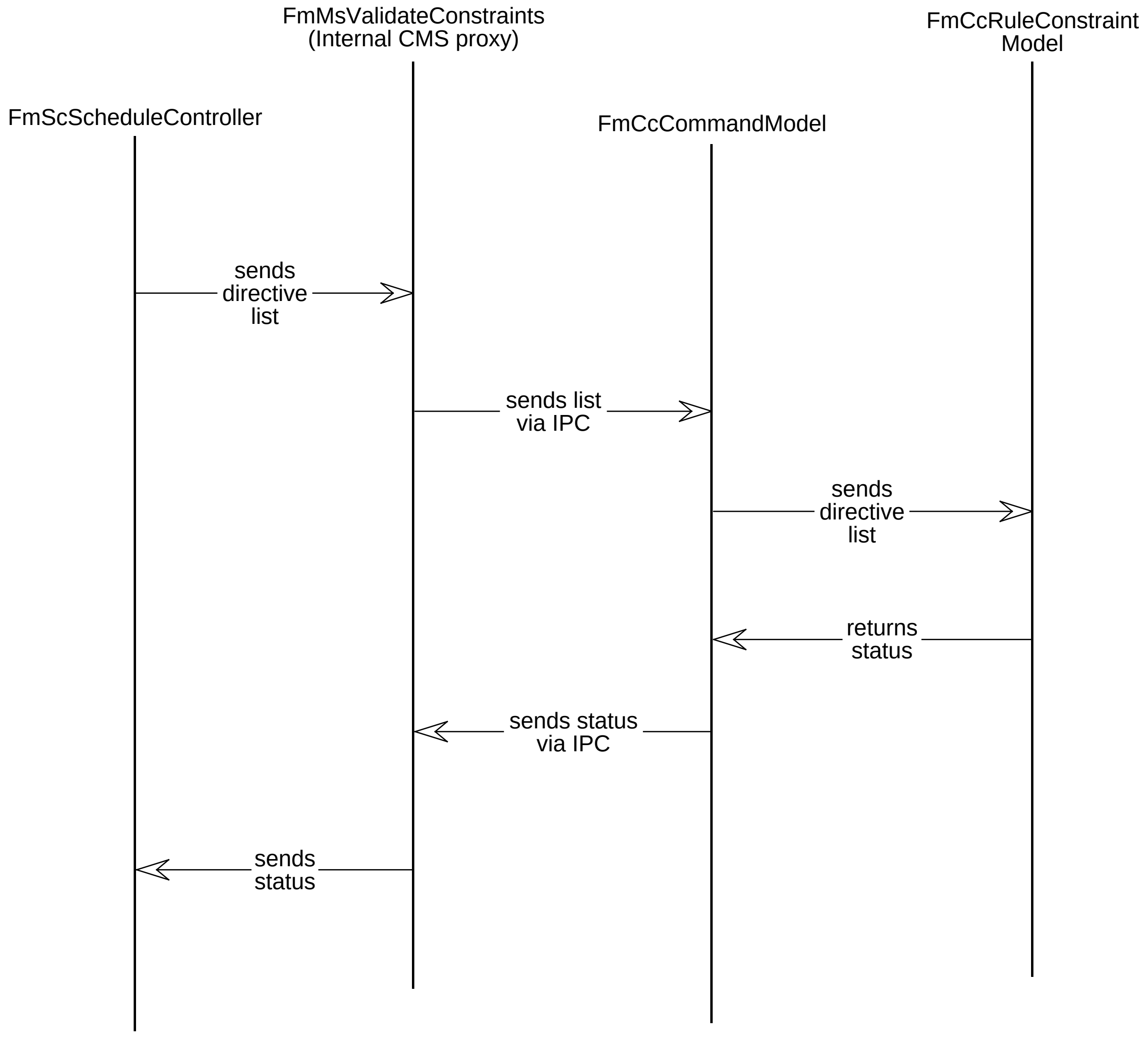


Figure 3.4-10 . Command Model Expanded Directive List Event Trace

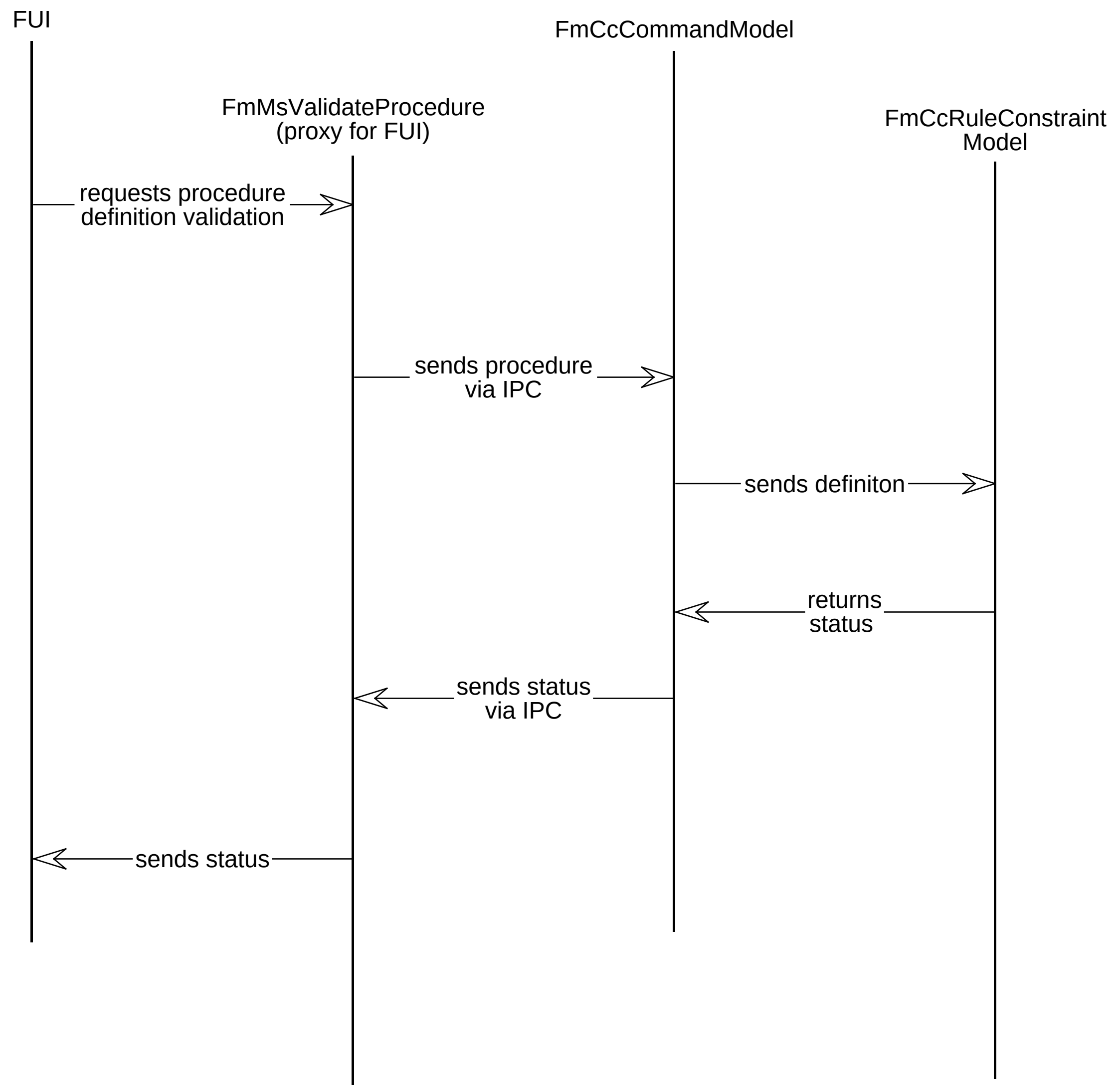


Figure 3.4-11. Command Model Command Procedure Receipt Event Trace

3.4.4.3 Command Model Command Procedure Receipt Scenario

3.4.4.3.1 Command Model Command Procedure Receipt Scenario Abstract

The Command Procedure Receipt scenario describes the receipt and processing of a procedure from FUI via FmMsValidateProcedure proxy (see Figure 3.4-11). The proxy opens the procedure file and restores a FoClProcedure class which is sent via IPC to FmCcCommandModel. The Command Model builds FmCcDirectiveLists from the procedure. The directive lists are sent to

the Rule Constraint model where the database defined rules associated with the directives are validated. An FoMsCMSStatus result is returned to FUI.

3.4.4.3.2 Command Model Command Procedure Receipt Summary Information

Interfaces:

- FUI

Stimulus:

- Receipt of Procedure

Desired Response:

- Status returned

Pre-Conditions:

- FUI procedure builder has been initiated
- Command model software has been initiated

Post-Conditions:

- None

3.4.4.3.3 Command Model Command Procedure Receipt Scenario Description

The Command Model receives the procedure from FUI via the FoMsValidateProcedure proxy. It creates FmCcDirectiveLists for each possible path in a procedure. If another procedure is referenced within the procedure being validated, it too is expanded. These lists are each sent to the Rule Constraint Model for command rule-based constraint validation. Each call to the Rule Constraint Model returns a FoMsCMSStatus. The Command Model combines the Conflicting command information contained within the FoMsCMSStatus for each directive list that was built for the procedure into one FoMsCMSStatus which is returned to FUI.

3.4.4.4 Command Model Activity Definition List Receipt Scenario

3.4.4.4.1 Command Model Activity Definition List Receipt Scenario Abstract

The Activity Definition Receipt scenario describes the receipt and processing of activity definitions from DMS via the FmMsValidateActivities proxy (see Figure 3.4-12).

3.4.4.4.2 Command Model Activity Definition List Receipt Summary Information

Interfaces:

- DMS

Stimulus:

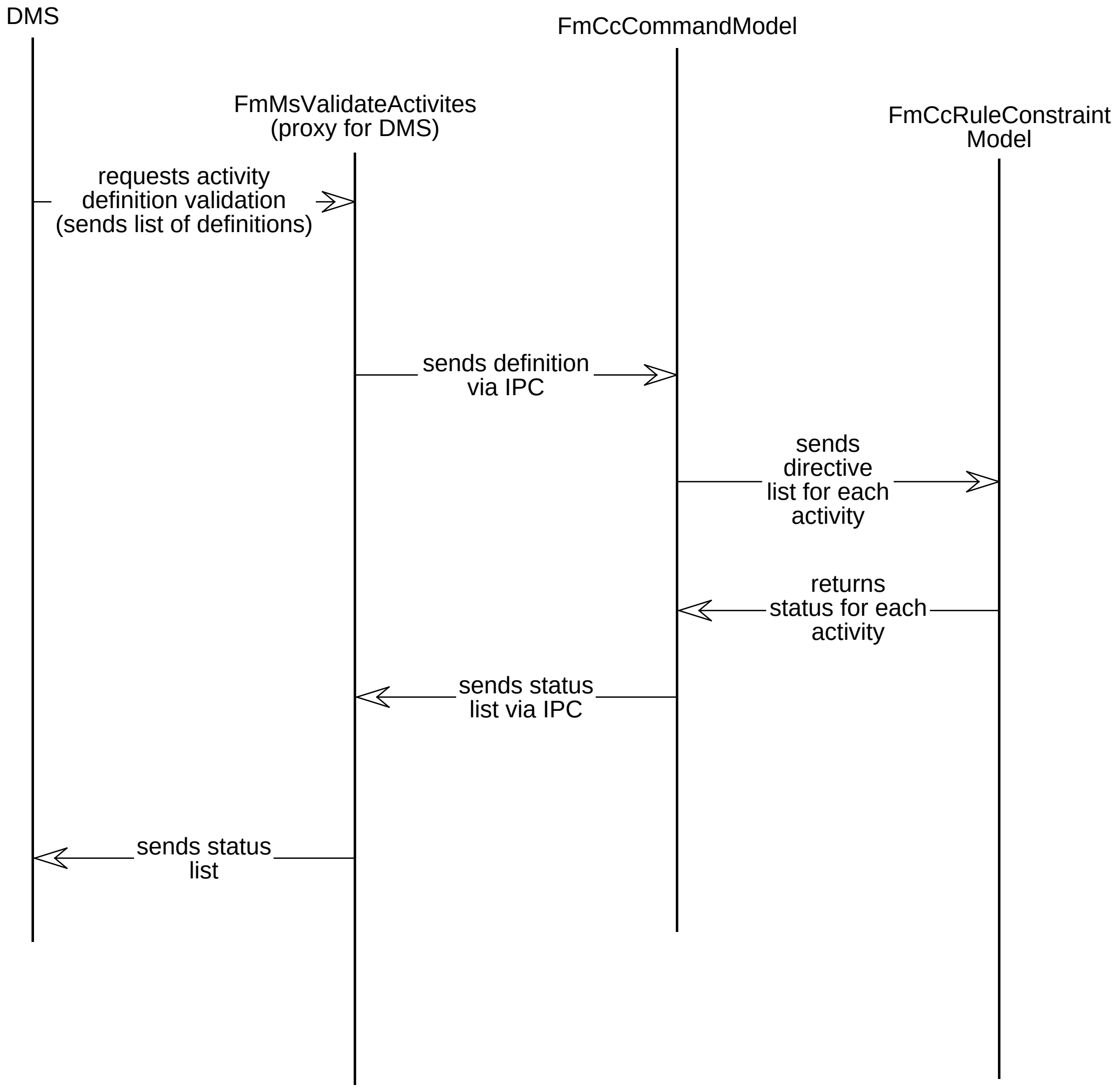

DMS
FmCcCommandModel
FmMsValidateActivites (proxy for DMS)
FmCcRuleConstraint Model
requests activity definition validation (sends list of definitions)
sends definition via IPC
sends directive list for each activity
returns status for each activity
sends status list via IPC
sends status list

Figure 3.4-12. Command Model Activity Definition List Receipt Event Trace

- Receipt of Activity Definition List

Desired Response:

- Status returned

Pre-Conditions:

- DMS validation software has been initiated
- Command model software has been initiated

Post-Conditions:

- None

3.4.4.4.3 Command Model Activity Definition List Receipt Scenario Description

The Command Model receives a list of activity definitions from DMS via the FoMsValidateActivities proxy. It creates FmCcDirectiveList for each activity. Since activities may reference procedures, the definitions are expanded. Multiple directive lists may exist for each activity due to procedure expansion described above. These lists are each sent to the Rule Constraint Model for command rule-based constraint validation. Each call to the Rule Constraint Model returns a FoMsCMSStatus. The Command Model combines the Conflicting command information contained within the FoMsCMSStatus for each directive list that was built for an activity and creates one FoMsCMSStatus for that activity. A list of FoMsCMSStatuses, one for each activity being command rule-based constraint checked, is returned to DMS.

3.4.4.5 Command Model RTS Load Definition Receipt Scenario

3.4.4.5.1 Command Model RTS Load Definition Receipt Scenario Abstract

The RTS Definition Receipt scenario describes the receipt and processing of an RTS load contents from CMS Load Catalog (FmLdLoadCatalog) via FmMsValidateConstraints proxy (see Figure 3.4-13).

3.4.4.5.2 Command Model RTS Load Definition Receipt Summary Information

Interfaces:

- CMS Load Catalog
- DMS

Stimulus:

- Receipt of RTS Definition

Desired Response:

- Status returned

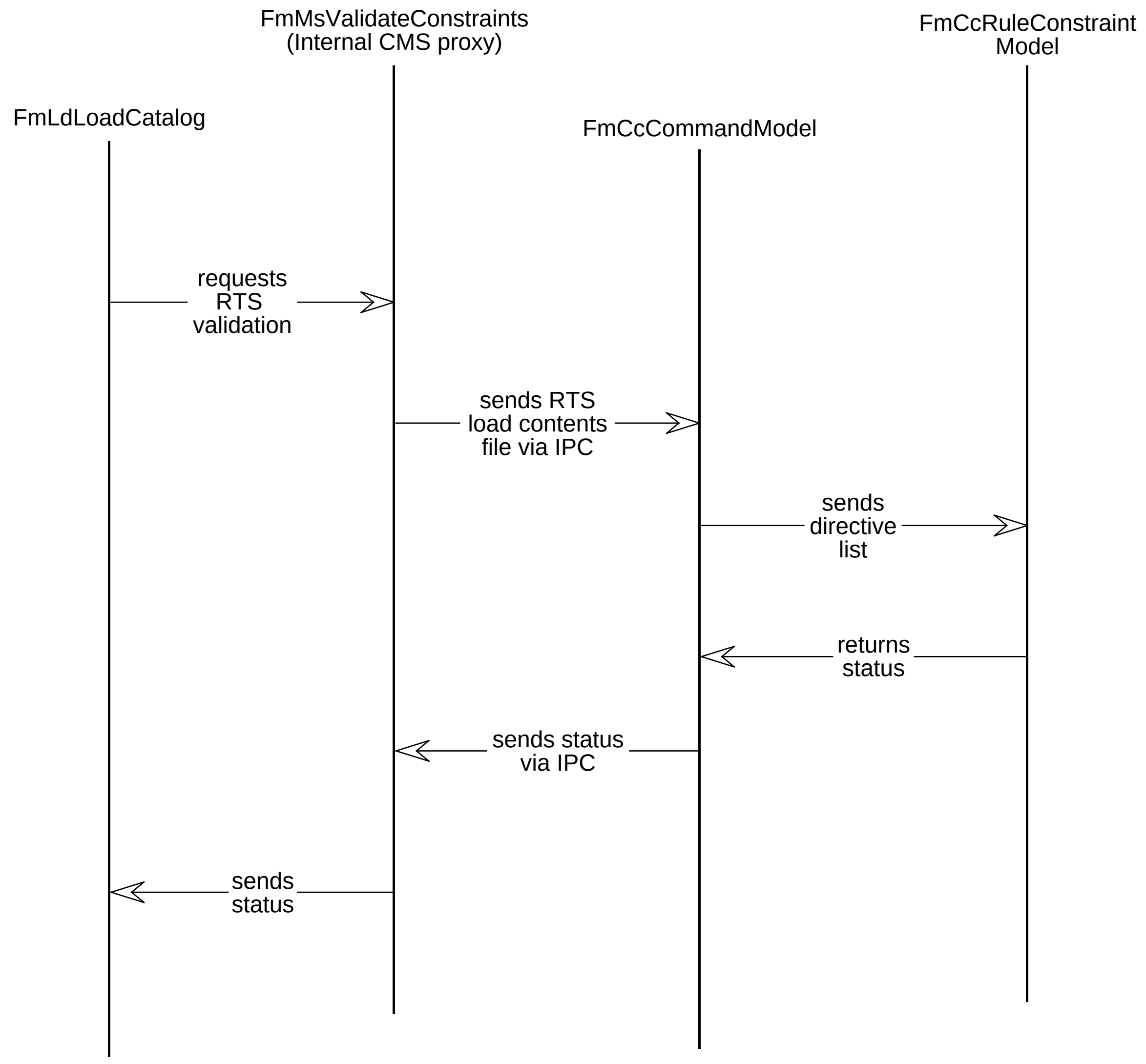


Figure 3.4-13. Command Model RTS Load Definition Receipt Event Trace

Pre-Conditions:

- Load catalog software has been initiated
- Command model software has been initiated

Post-Conditions:

- None

3.4.4.5.3 Command Model RTS Load Definition Receipt Scenario Description

The Command Model receives the RTS load contents definition from Load catalog via FmMsValidateConstraints. It builds an FmCcDirectiveList and sends the list to the Rule Constraint Model for command rule-based constraint validation. An FoMsCMSStatus is returned to the Load Catalog

3.4.5 Command Model Data Dictionary

Preprocessor Macros

`_FmCcCommandModel_h_`

Types

`class` **`FmCcCommandModel`**

The command model provides access to command level rule based constraint checking for scheduled commands, commands within an RTS load contents definition file, commands within activity definitions, and commands within procedures.

Public Functions

`EcTInt` **`BuildActivityList`**`(const FoAcActivityDef&, const RWSlistCollectables&)`

Builds the lists for Activities

`EcTInt` **`BuildProceduresLists`**`(const FoClProcedure&, const RWSlistCollectables&)`

Builds the lists for procedures

`EcTVoid` **`HandleMessage`**`(const RWCollectable&)`

Handles all messages between proxies. The proxies are as follows: FmMsValidateConstraints - used internal to CMS, FmMsValidateProcedure - used by FUI, and FmMsValidateActivities - used by DMS

`FoMsCMSStatus&` **`ProcessActivityDefinitions`**`(const FoFdActivityList&)`

For each activity in the FoFdActivityList the commands within the activity are sent to the FmCcRule ConstraintModel to be validated

`FoMsCMSStatus&` **`ProcessCommandList`**`(const FmScConstCk&)`

Process scheduled directives received from PAS

`EcTVoid` **`ProcessIF`**`(RWSlistCollectablesIterator&, RWSlistCollectables&)`

Processes IF directive when expanding procedures

`FoMsCMSStatus&` **`ProcessProcedureDefinitions`**`(const FoClProcedure&)`

Validates Procedures

`FoMsCMSStatus&` **`ProcessRTSDefinitions`**`(const FoDirectiveList&)`

Validates RTS load contents files

`RWSlistCollectable&` **`ProcessSTART`**`(RWCString&, RWCString&)`

Processes START directive to expand procedures

`EcTVoid` **`SendConflictInfo`**`(const FoMsCMSStatus&)`

Sends a single FoMsCMSStatus - results of rule based constraint checking to either FUI, the scheduled controller (CMS), or the load catalog (CMS)

`EcTVoid` **`SendStatuses`**`(const RWSlistCollectables&)`

Sends a list of FoMsCMSStatus(es) to DMS

Preprocessor Macros

_FmCcCommandRule_h_

Types

enum **FlagType**

Enumeration type for hard/soft flag

Enumerators

H
S

class **FmCcCommandRule**

Public Functions

`GetHardSoftFlag(void)`

gets myHardSoftFlag

`RWCString&` **`GetTrigger`**`(void)`

Gets myTrigger

`EcTVoid` **`SetHardSoftFlag`**`(const enum)`

Sets myHardSoftFlag

`EcTVoid` **`SetTrigger`**`(const RWCString&)`

sets myTrigger

Private Data

`FlagType` **`myHardSoftFlag`**
`RWCString` **`myTrigger`**

Include Files

`FmCcOffsetRule.h`

Preprocessor Macros

_FmCcNoCmdsAfterRule_h_

Types

class **FmCcNoCmdsAfterRule**

This class determines if any command is performed after the offset. The rule reads as follows:

If command A then NO commanding at least X seconds before command A.

Base Classes

`public` **`FmCcOffsetRule`**

Public Functions

`FoMsConflictInfo&` **`Evaluate`**`(const RWDlistCollectables&, EctInt)`

Test to see if the next command is issued before the offset

Include Files

`FmCcOffsetRule.h`

Preprocessor Macros

`_FmCcNoCmdsBeforeRule_h_`

Types

`class` **`FmCcNoCmdsBeforeRule`**

this class determins if any command is performed between the offset and the current command. The rule reads as follows:

If command A then NO commanding at least X seconds before command A.

Base Classes

`public` **`FmCcOffsetRule`**

Public Functions

`FoMsConflictInfo&` **`Evaluate`**`(const RWDlistCollectables&, EcTInt)`

Include Files

`FmCcCommandRule.h`

Preprocessor Macros

`_FmCcNoExistRule_h_`

Types

`class` **`FmCcNoExistRule`**

This rule searches the entire command list to determine if myExcluder is in the command list If found it is a violation. The rule reads as follows:

If command A occurs then command B must not occur

Base Classes

`public` **`FmCcCommandRule`**

Public Functions

`FoMsConflictInfo&` **`Evaluate`**`(const RWDlistCollectables&)`

Determines if myExcluder is in the command list

Protected Functions

`RWCString&` **`GetExcluder`**`(void)`

gets myExcluder

`EcTVoid` **`SetExcluder`**`(const RWCString&)`

Sets myExcluder

Private Data

`RWCString` **`myExcluder`**

The command mnemonic that is not to occure in the command list

Include Files

`FmCcOffsetRule.h`

Preprocessor Macros

`_FmCcNoRTCmdsRule_h_`

Types

`class` **`FmCcNoRTCmdsRule`**

This class ensures that commands are not being sent to the same RT at the current (executing) command before the offset expires

Base Classes

`public` **`FmCcOffsetRule`**

Public Functions

`FoMsConflictInfo&` **`Evaluate`**`(const RWDlistCollectables&, const EcTInt)`

Ensures that once a command is sent to the RT no other commands are sent to that RT before the offset

Include Files

`FmCcCommandRule.h`

Preprocessor Macros

`_FmCcOffsetRule_h_`

Types

`class` **`FmCcOffsetRule`**

This is the base class for those rules that have offset times: FmCcRepeatAfterRule, FmCcNoCmdsBeforeRule, FmCcNoCmdsAfterRule, and FmCcNoRTCmdsRule

Base Classes

`public` **`FmCcCommandRule`**

Public Functions

`virtual FoMsConflictInfo&` **`FmCcOffsetRule::Evaluate`**`()`

Abstact function for evaluating offset rules - no processing takes place

Protected Functions

`EcTULongInt` **`GetOffset`**`(void)`

gets myOffset

`EcTvoid` **`SetOffset`**`(const EcTULongInt)`

sets myOffset

Private Data

`EcTULongInt` **`myOffset`**

Offset for commanding - represents seconds

Include Files

`FmCcCommandRule.h`

Preprocessor Macros

`_FmCcPostRule_h_`

Types

`class` **`FmCcPostRule`**

The Post rule ensures that if a certain command is issued then another specified command must occur within a given time frame. The rule reads as follows:

If command A occurs, then command B must occur an dmust be at least X time later, and at most Y time later. A is the trigger and B is the satisfier.

Base Classes

`public` **`FmCcCommandRule`**

Public Functions

`FoMsConflictInfo&` **`Evaluate`**`(const RWDlistCollectables&, EcTInt)`

Evaluates the PostRule: ensures that if a certain command is issued then another specified command must occur within a given time frame.

Protected Functions

`EcTULongInt` **`GetMaxTime`**`(void)`

gets myMaxTime

`EcTULongInt` **`GetMinTime`**`(void)`

gets myMinTime

`FmCcSymbolDef&` **`GetPacifier`**`(void)`

gets myPacifier

`EcTVoid` **`SetMaxTime`**`(const EcTULongInt)`

sets myMaxTime

`EcTVoid` **`SetMinTime`**`(const EcTULongInt)`

Sets myMinTime

`EcTVoid` **`SetPacifier`**`(const FmCcSymbolDef&)`

sets myPacifier

Private Data

`EcTULongInt` **`myMaxTime`**

time that the pacifier can be found in the command list

`EcTULongInt` **`myMinTime`**

the pacifier can be found in the command list

`FmCcSymbolDef` **`myPacifier`**

symbols defined. If there are multiple symbols defined then any one of the symbols may satisfy the rule.

Include Files

`FmCcCommandRule.h`

Preprocessor Macros

`_FmCcPreRule_h_`

Types

`class` **`FmCcPreRule`**

The preRule ensures that specified commands are executed before the command being evaluated. The rule reads as follows:

If command A occurs, then command B must have occurred at least X time earlier, and at most Y time earlier. Command C must not occur between B and A. A is the trigger, B is the satisfier and C is the excluded which is optional

Base Classes

`public` **`FmCcCommandRule`**

Public Functions

`FoMsConflictInfo&` **`Evaluate`**`(const RWDlistCollectables&, EcTInt)`

Evaluates the PreRule: ensures that specified commands are executed before the command being evaluated. It allows the exclusion of a specified command.

Protected Functions

`FmCcSymbolDef&` **`GetExcluder`**`(void)`

gets myExcluder

`EcTULongInt` **`GetMaxTime`**`(void)`

get myMaxtime

`EcTULongInt` **`GetMinTime`**`(void)`

gets myMinTime

`FmCcSymbolDef&` **`GetSatisfier`**`(void)`

gets mySatisfier

`EcTVoid` **`SetExcluder`**`(const FmCcSymbolDef&)`

Sets myExcluder

`EcTVoid` **`SetMaxTime`**`(const EcTULongInt)`

Sets myMaxTime

`EcTVoid` **`SetMinTime`**`(const EcTULongInt)`

sets myMinTime

`EcTVoid` **`SetSatisfier`**`(const FmCcSymbolDef&)`

sets mySatisfer

Private Data

`FmCcSymbolDef` **`myExcluder`**

definition may be defined for it

`EcTULongInt` **`myMaxTime`**

time that the satisfier can be found. It is optional, if it is not set then the max time is the beginning of the command list.

`EcTULongInt` **`myMinTime`**

that the satisfier can be found.

`FmCcSymbolDef` **`mySatisfier`**

symbols defined. If there are multiple symbols defined then any one of the symbols may satisfy the rule.

Include Files

`FmCcOffsetRule.h`

Preprocessor Macros

`_FmCcRepeatAfterRule_h_`

Types

`class` **`FmCcRepeatAfterRule`**

This rule ensures that the command is not repeated before the offset expires. The command rule reads as follows:

If Command A then command A not before X seconds

Base Classes

`public` **`FmCcOffsetRule`**

Public Functions

`FoMsConflictInfo&` **`Evaluate`**`(const RWDlistCollectables&, EcTInt)`

Test to see if the trigger command is repeated before the offset expires

Preprocessor Macros

`_FmCcRuleConstraintModel_h_`

Types

`class` **`FmCcRuleConstraintModel`**

An instance of this class is instantiated for each command list that will be command rule-based constraint checked

Public Functions

`GetStatus(void)`

gets myStatus

`RWDlistCollectables&` **`GetCommandList`**`(void)`

gets myCommandList

`RWSlistCollectables&` **`GetConstraintList`**`(void)`

gets myConstraintList

`EcTVoid` **`PerformConstraintCheck`**`(const FmCcCommandRule&)`

Given a specific rule it evaluetes the command to verify that the constraint is not being violated

`FoMsCMSStatus&` **`ProcessList`**`(void)`

Steps through the mycommandList, verifies that no more than 8 commands are issued per second. Calls PerformConstraintCheck for those commands that have rules associated with them

`EcTVoid` **`SetCommandList`**`(const RWDlistCollectables&)`

sets myCommandList

`EcTVoid` **`SetConstraintList`**`(const RWSlistCollectables&)`

sets myConstraintList

`EcTVoid` **`SetStatus`**`(const enum&)`

Sets myStatus

Private Data

`RWDlistCollectables` **`myCommandList`**

command list to be constraint checked

`RWSlistCollectables` **`myConstraintList`**

list of CommandInfo to put in FoMsCMSStatus

`FoMsCMSStatus` **`myStatus`**

enumerated - either pass, fail, or pending. Depending on the value of myStatus the correct FoMsCMSStatus will be created.

Include Files

`FmCcCommandRule.h`

Preprocessor Macros

`_FmCcScalarRule_h_`

Types

`enum` **`CompareType`**

Enumeration for comaparison type

Enumerators

`EQ`
`GE`
`GT`
`LE`
`LT`
`NE`

`template<class Type> class` **`FmCcScalarRule`**

This class is a template class. The scalar rule can be performed on a single data field. A separate rule must be defined for each data field that the rule will be applied. Data fields may be of type int, double, or float ...(??) DFCD should state valid types. The rule reads as follows:

If command A occurs, the value of a particular data field must be less than, or greater than, or equal to, or less than or equal to, or greater than or equal to, or not equal to X.

Base Classes

`public` **`FmCcCommandRule`**

Public Functions

`FoMsConflictInfo&` **`Evaluate`**`(const RWSlistCollectables&)`

list for the command and performs the proper operation.

Protected Functions

`GetOperator(void)`

Gets operation to perform

`Type` **`GetComparisonValue`**`(void)`

Gets ComparisonValue

`EcTInt` **`GetDataField`**`(void)`

Gets data field to compare

`RWCString&` **`GetSubfieldName`**`(void)`

Gets Parameter name that test is being performed on

`EcTVoid` **`SetComparisonValue`**`(const EcTInt)`

Sets myComparisonValue

`EcTVoid` **`SetDataField`**`(const EcTInt)`

Sets myDataField

`EcTVoid` **`SetOperator`**`(const enum)`

Sets myOperator

`EcTVoid` **`SetSubfieldName`**`(const RWCString&)`

Sets mySubfieldName

Private Data

`Type` **`myComparisonValue`**

The value to test the command parameter against

`EcTInt` **`myDataField`**

The data field to test

`CompareType` **`myOperator`**

the operation to perform

`RWCString` **`mySubfieldName`**

The parameter mnemonic

Include Files

`FmCcCommandRule.h`

Preprocessor Macros

`_FmCcTelemetryRule_h_`

Types

`class` **`FmCcTelemetryRule`**

This rule allows a specific warning message to be output to the user when a command is encountered. The Hard/Soft flag for this rule should be set to S(oft). The rule reads as follows:

If A occurs, print an FOT-supplied rule-specific text string in the conflict report

Base Classes

`public` **`FmCcCommandRule`**

Public Functions

`FoMsConflictInfo&` **`Evaluate`**`(void)`

Returns myText to FmCcRuleConstraintModel

Protected Functions

`RWCString&` **`GetText`**`(void)`

Gets myText

`EcTVoid` **`SetText`**`(const RWCString&)`

sets myText

Private Data

`RWCString` **`myText`**

FOT supplied text message

Preprocessor Macros

`_FmMsValidateActivities_h_`

Types

`class` **`FmMsValidateActivities`**

This class is a proxy class for DMS to send one or more activities to be constraint checked

Public Functions

`EcTInt` **`CreateConnection`**`(void)`

Establishes a connection with FmCcCommandModel to permit the transfer of a list of activity definitions to be command rule-based constraint checked.

`EcTVoid` **`DestroyConnection`**`(void)`

Destroys the connection with FmCcCommandModel

`RWSlistCollectables&` **`Receive`**`(void)`

Receives the results of rule-base command constraint checking, in regards to activities this is a list of FoMsCMSStatus(es)

`RWSlistCollectables&` **`Send`**`(const FoFdActivityList&)`

Sends a list of one or more activity definitions to the FmCcCommandModel for rule-base command constraint checking

`RWSlistCollectables&` **`ValidateActivities`**`(const FoFdActivityList&)`

Requests that a list of activities be rule based constraint checked This is the call used by the service user (DMS)

Preprocessor Macros

`_FmMsValidateConstraints_h_`

Types

`class` **`FmMsValidateConstraints`**

This class represents the interface proxy class between CMS internal subsystems and the FmCcCommandModel class. FmC-cCommandModel manages the command rule-based constraint checking.

Public Functions

EcTInt **CreateConnection**(void)

Establishes a connection with FmCcCommandModel to receive constraint checking request from the schedule controller and the load catalog

EcTVoid **DestroyConnection**(void)

Destroys the connection with FmCcCommandModel

FoMsCMSStatus& **Receive**(void)

Receives the results of rule-base command constraint checking, FoMsCMSStatus

FoMsCMSStatus& **Send**(const RWCollectable&)

Sends either a FmScConstCk command list from the schedule controller or a FoCcDirective list created from an RTS load contents file to the FmCcCommandModel for rule-base command constraint checking

FoMsCMSStatus& **ValidateCommands**(const FmScConstCk&)

FmScScheduleController invokes this function to send the DAS scheduled command list to be command rule-based constraint checked

FoMsCMSStatus& **ValidateRTS**(const RWCString&, const RWCString&)

FmLdLoadCatalog invokes this function to send the directory name and load name from the generate RTS load request to be command rule-based constraint checked. This function creates the FoCcDirectiveList to the FmCcCommandModel.

Preprocessor Macros

_FmMsValidateProcedure_h_

Types

class **FmMsValidateProcedure**

This class represents the interface proxy class between FUI subsystem and the CMS FmCcCommandModel class. FmCcCommandModel manages the command rule-based constraint checking .

Public Functions

EcTInt **CreateConnection**(void)

Establishes a connection with FmCcCommandModel to permit the transfer of procedure validation request

EcTVoid **DestroyConnection**(void)

Destroys the connection with FmCcCommandModel

FoMsCMSStatus& **Receive**(void)

Receives the results of rule-base command constraint checking, FoMsCMSStatus

FoMsCMSStatus& **Send**(const FoClProcedure&)

Sends a FoClProcedure object to the FmCcCommandModel for rule-base command constraint checking

FoMsCMSStatus& **ValidateProcedure**(const FoMsValidateProcReq&)

FUI invokes this function to send a validate procedure request to the FmCcCommandModel, so that the procedure can be command rule-based constraint checked

Include Files

`FoEcGroundDirective.h`

Preprocessor Macros

`_FoEcComment_h_`

Types

`class` **`FoEcComment`**

This class is used for comments in the ground script It may represent a space directive or simply plain text

Base Classes

`public` **`FoEcGroundDirective`**

Private Data

`RWCString` **`myText`**

The string to hold the comment

Include Files

`FoEcTime.h`

Preprocessor Macros

`_FoEcDeltaTime_h_`

Types

`class` **`FoEcDeltaTime`**

This time is derived from FoEcTime it represents a delta time that is plus/minus so many minutes and seconds from some specified time

Base Classes

`public` **`FoEcTime`**

Private Data

`EcTChar` **`myPlusMinusSign`**

the plus/minus sign indicates whehter time should be added to or taken from the specified time

`EcTChar` **`myStartStopIndicator`**

the start/stop indicator is used to determine if the delta should be taken from the start time or the stop time specified The default is the start time

Preprocessor Macros

`_FoEcDirective_h_`

Types

`class` **`FoEcDirective`**

This is a shared class in the FOS. It represents a directive in the system. The directive can be used in an Activity definition, a CMS schedule of directives, a command procedure and in a ground script.

Public Functions

void **CheckSyntax**(EcTInt errcode)
void **Execute**(void)
void **LogDirective**(void)
void **Parse**(void)
void **UpdateStatus**(void)

Private Data

EcTInt **myActivityId**

The activity id in which this directive is scheduled

EcTInt **myDASId**

The DAS Id in which the directive is scheduled

FuTdDataSource* **myDataSourceId**

RWCString **myDirectiveText**

The directive text

FuGsGroundScriptControl* **myGndScript**

EcTInt **myLineNum**

the line number in the procedure

RWSlistCollectables **myParameters**

the parameters of a directive, the parameters ma the subfield parameters of a space or real-time command or parameters of a ECL keyword (limits "on")

FoClProcedure* **myProc**

The procedure in which the directive was written

FuClProcControlWin* **myProcControl**

enum **myProcFlag**

enum **mySource**

EcTInt **myStatus**

Directive status

Private Types

enum

The source of the directive: manual, procedure, or ground script

Enumerators

gs
manual
proc

enum

a flag indicating if the directive is in a procedure

Enumerators

n
y

Include Files

`FoEcDirective.h`

Preprocessor Macros

`_FoEcGroundDirective_h_`

Types

`class` **`FoEcGroundDirective`**

This class is derived from FoEcDirective is represents a directive that is placed in the ground script and is issued from the ground system, that is this type of directive is never found on the spacecraft

Base Classes

`public` **`FoEcDirective`**

Private Data

`RWCString` **`myKeyword`**

Represents the keyword for the ground directive

Include Files

`FoEcGroundDirective.h`

Preprocessor Macros

`_FoEcLabel_h_`

Types

`class` **`FoEcLabel`**

This class is a ground directive that represents a label in a procedure. It allows the user to "goto" the label

Base Classes

`public` **`FoEcGroundDirective`**

Public Functions

`RWCString&` **`GetName`**`(void)`

Retrieves myName

`EcTInt` **`GetOffset`**`(void)`

Retrieves offset

`EcTVoid` **`Jump`**`(void)`

Jumps to offset or label name location

`EcTVoid` **`SetName`**`(const RWCString&)`

sets myName

`EcTVoid` **`SetOffset`**`(EcTInt)`

Sets myOffset

Private Data

`RWCString` **`myName`**

Label Name

`EcTInt` **`myOffset`**

Label offset in the procedure

Include Files

`FoEcGroundDirective.h`

Preprocessor Macros

`_FoEcRTCommand_h_`

Types

`class` **`FoEcRTCommand`**

This class is derived from FoEcGroundDirective. It represents a real-time command. Real-time Commands are commands to the spacecraft that are issued from the ground system.

Base Classes

`public` **`FoEcGroundDirective`**

Public Functions

`RWBitVec&` **`GetBinary`**`(void)`

Retrieves the binary representation of the command

`RWCString&` **`GetMnemonic`**`(void)`

retrieves the data base defined command mnemonic of the directive

`EcTInt` **`SetBinary`**`(void)`

Sets the binary representation for the directive

`EcTVoid` **`SetMnemonic`**`(const RWCString&)`

Sets the command mnemonic for the directive

Private Data

`RWBitVec` **`myBinary`**

The binary representation of the command mnemonic

`RWCString` **`myMnemonic`**

the command mnemonic

Include Files

`FoEcDirective.h`

Preprocessor Macros

`_FoEcSpaceDirective_h_`

Types

`class` **`FoEcSpaceDirective`**

This class is derived from FoEcDirective. It represents a space command. Space Commands are commands to the spacecraft that are issued from the spacecraft. These commands are either ATC or RTS commands or instrument commands.

Base Classes

`public` **`FoEcDirective`**

Public Functions

`EcTInt` **`FigureBinary`**`()`

Generates the binary representation of this directive.

`RWBitVec&` **`GetBinary`**`()`

Returns the binary.

`EcTInt` **`GetInhibitId`**`()`

Returns the inhibit id.

`RWCString&` **`GetMnemonic`**`()`

Returns the mnemonic.

`EcTInt` **`GetRTSFlag`**`()`

Returns the RTS flag.

`EcTVoid` **`SetInhibitId`**`(EcTInt)`

Sets the inhibit id to the input value.

`EcTVoid` **`SetMnemonic`**`(const RWCString&)`

Sets the mnemonic to the input value.

`EcTVoid` **`SetRTSFlag`**`(EcTInt)`

Sets the RTSFlag to the input value.

Private Data

`RWBitVec` **`myBinary`**

The binary representation of this directive, the format of which is described by the ICD.

`EcTInt` **`myInhibitId`**

The group id indicating what resource this directive could have an effect on.

`RWCString` **`myMnemonic`**

The mnemonic for the directive.

`EcTInt` **`myRTSFlag`**

An indicator specifying if this directive is part of an RTS or not.

Include Files

`FoEcTime.h`

Preprocessor Macros

`_FoEcSpaceTime_h_`

Types

`class` **`FoEcSpaceTime`**

This class represent the time for the commands on the spacecraft. Time is converted so that when a command `is said to execute every second is is actually every 1.024` seconds.

Base Classes

`public` **`FoEcTime`**

Public Functions

`EcTFloat` **`GetConversionFactor`**`()`

Returns the conversion factor.

`EcTVoid` **`SetConversionFactor`**`(EcTFloat)`

Sets the conversion factor to the input value.

Private Data

`EcTFloat` **`myConversionFactor`**

The numeric factor which converts actual seconds to spacecraft seconds.

Include Files

`RWTime.h`

Preprocessor Macros

`_FoEcTime_h_`

Types

`class` **`FoEcTime`**

this class is generic for FOS so that all times are derived from the same epoch

Base Classes

`public` **`RWTime`**

Public Functions

`RWTime&` **`GetEpoch`**`()`

Returns the epoch.

`EcTVoid` **`SetEpoch`**`(const RWTime&)`

Sets the epoch to the input time.

Private Data

RWTime **myEpoch**

The epoch upon which the time is based. Time is computed as the number of seconds since the epoch. The default epoch for RWTime is Jan 1, 1901 at 00:00:00.

Preprocessor Macros

_FoMsCMSStatus_h_

Types

class **FoMsCMSStatus**

This class is used to return processing status to CMS's external interfaces. It returns status for constraint checking and for load generation.

Private Data

EcTInt **myId**

or the Instruction request id that the status is in response to

RWCString **myStatus**

The status is either:

```
complete - everything processed without error
pending -  the constraint check was complete with
           soft constraints only
failed - constraint violations found were hard constraints
         load generation failed
```

Preprocessor Macros

_FoMsConflictInfo_h_

Types

class **FoMsConflictInfo**

This class gives the identifying information on constraint violations. It specifies the id, the command mnemonic, the conflicting command, the time the constraint violation occurred, whether the violation is hard or soft and a textual description of the violaion

Private Data

RWCString **myCmdMnemonic**

the directive command mnemonic being constraint checked

RWCString **myConflictingCmd**

the command that violates the constraint rule

RWTime **myConstraintTime**

the time of the constraint

EcTInt **myId**

the ID represent different things for different constraint checking requests:

```
- the activity id of a command in a schedule
- the line number of a command in a procedure
- the buffer location of a command in an RTS load
  contents file
- the PDB activity definition id
```

`EcTInt` **`mySoftHardFlag`**

Indicates if the violatin is hard or soft

`RWCString` **`myViolationInfo`**

Textual description of the violation for messaging

Include Files

`FoMsCMSStatus.h`

Preprocessor Macros

`_FoMsStatusComplete_h_`

Types

`class` **`FoMsStatusComplete`**

Represents a good status from constraint checking or load generation

Base Classes

`public` **`FoMsCMSStatus`**

Include Files

`FoMsCMSStatus.h`

Preprocessor Macros

`_FoMsStatusFailed_h_`

Types

`class` **`FoMsStatusFailed`**

Represents a failed status from constraint checking of load generation

Base Classes

`public` **`FoMsCMSStatus`**

Include Files

`FoMsCMSStatus.h`

Preprocessor Macros

`_FoMsStatusPending_h_`

Types

`class` **`FoMsStatusPending`**

Represents a pending status from constraint checking This means that the constraint violations found are all soft constraints. If CMS is processing a DAS, we wait to continue processing of the load until a response is received from planning and scheduling. If CMS is processing an RTS load we wait for a response from FUI to continue processing the RTS load.

Base Classes

public **FoMsCMSStatus**

Include Files

FoUiInstruction.h

Preprocessor Macros

_FoMsValidateProcReq_h_

Types

class **FoMsValidateProcReq**

This message allows the user to request command rule-based constraint checking validation for a procedure. The user, FUI, fills in the necessary information and send the class to the FmMsValidateProcedure proxy.

Base Classes

public **FoUiInstruction**

Protected Functions

RWCString& **GetDirectory**(void)

gets myDirectory

RWCString& **GetProcName**(void)

gets myProcName

EcTVoid **SetDirectory**(const RWCString&)

sets myDirectory

EcTVoid **SetProcName**(const RWCString&)

sets myProcName

Private Data

RWCString **myDirectory**

directory where the procedure for constraint checking is located

RWCString **myProcName**

procedure name

3.5 Spacecraft Model

The Spacecraft Model is a persistent process running on the FOS Data Server. It models several aspects of spacecraft memory, including the ATC buffer, RTS buffers, and data tables. Spacecraft Model also maintains a binary image of selected portions of spacecraft memory, including the ATC buffer, RTS buffers, data tables, and SCC flight software.

The ATC buffer model maintained by Spacecraft Model represents the state of the ATC buffer after a particular ATC load is uplinked. Spacecraft Model maintains one instance of the ATC buffer model for each ATC load that is generated by CMS. When an ATC load is successfully uplinked to the spacecraft, Spacecraft Model replaces its current ATC buffer model with the ATC buffer model that reflects the uplinked load. Each ATC buffer model consists of a list of commands including the absolute time tags and location within the ATC buffer for each command. The buffer model mimics the wraparound feature of the ATC buffer onboard the AM-1 spacecraft. When an ATC load is being generated, Spacecraft Model uses the ATC buffer model to determine appropriate partitions of the load based on the predicted contents of the ATC buffer following the previous load. The ATC buffer model is also used in generating ATC command-to-memory map reports and displays.

The RTS buffer model maintained by Spacecraft Model represents the state of the 128 RTS buffers onboard AM-1. The RTS buffer model consists of two parts: the RTS load-to-buffer map, which is a list of the loads most recently uplinked to the 128 buffers; and the RTS command-to-memory map, which has lists of commands currently loaded in each of the 128 RTS buffers. The RTS load-to-buffer map and command-to-memory map are used in generating the RTS memory map reports and displays.

The table model maintained by Spacecraft Model represents the state of the data tables onboard AM-1. In order for a table to be included in the table model, it must be defined in the FOS database. The table model consists of a list of tables that are defined in the database and, for each table in the list, the name of the load most recently uplinked to that table. The table model is used in generating the table map report.

Spacecraft Model maintains binary ground reference images of the ATC buffer, RTS buffers, data tables, and SCC flight software. Whenever a load to one of these areas of memory is uplinked successfully, the corresponding ground reference image is updated from the load image. Also, Spacecraft Model will update a ground reference image from a load image or dump image on request. The ground reference images are used in doing dump comparisons and in generating reports.

3.5.1 Spacecraft Model Context

The CMS Spacecraft Model interfaces with FOS User Interface subsystem, the Data Management subsystem, the CMS Schedule Controller task, and the CMS load catalog as shown in the Spacecraft Model Context Diagram (see Figure 3.5-1) and summarized below.

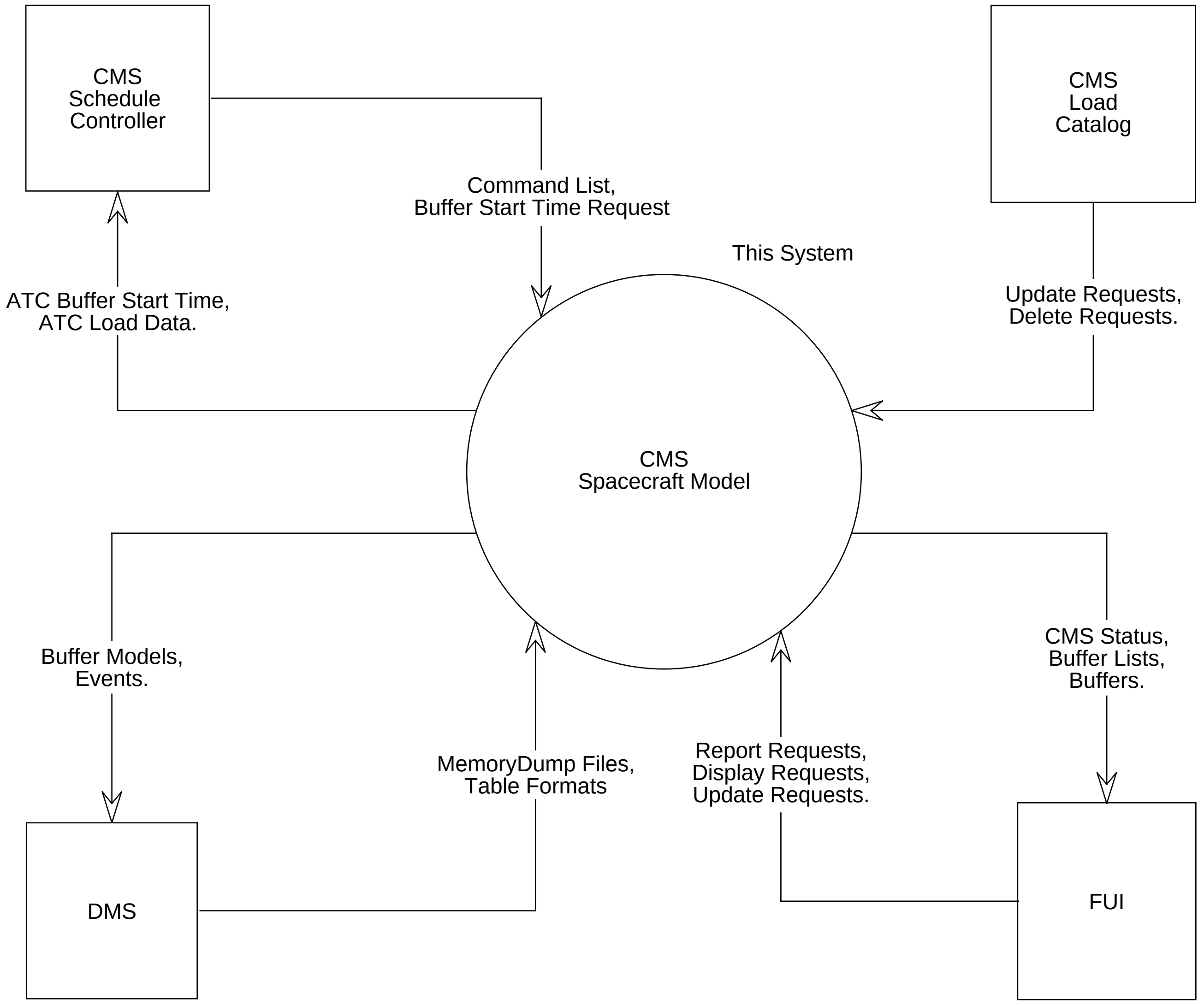

CMS Schedule Controller
CMS Load Catalog
Command List, Buffer Start Time Request
This System
ATC Buffer Start Time, ATC Load Data.
Update Requests, Delete Requests.
CMS Spacecraft Model
Buffer Models, Events.
CMS Status, Buffer Lists, Buffers.
MemoryDump Files, Table Formats
Report Requests, Display Requests, Update Requests.
DMS
FUI

Figure 3.5-1. Spacecraft Model ontext Diagram

CMS Schedule Controller:

- Requests the time of the first command in the most recent predicted buffer. Once constraint checking has passed, Schedule Controller sends the DAS directive list, the requested uplink window, the time of the first command used to make the load, and the DAS id.
- Receives a list of load data objects, each of which includes the directive list for the ATC load to be generated, the name of the load, and the updated uplink window for scheduling.
- Sends an update buffer request to update the buffer status to reflect successful load generation.

CMS Load Catalog:

- Sends a request to update the buffers from predicted to actual.
- In the event of a late change, load catalog sends a request to delete specified predicted ATC buffer models.

User Interface:

- Sends a request to retrieve a list of available ATC or RTS buffers for display. The user may then select a specific buffer for the ATC or RTS buffer displays.
- Requests the generation of the map reports for the ATC buffer, RTS buffers, and Tables, comparison reports, and image reports.
- Requests that the ground image be overwritten with a dump image.

Data Management:

- Receives Events, which are status messages about CMS Spacecraft Model processing
- Checkpoints the ATC and RTS buffers.
- DMS provides the memory dump file and the table formats to Spacecraft Model.

3.5.2 Spacecraft Model Interfaces

Table 3.5.2. Spacecraft Model Interfaces (1 of 3)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Get ATC Buffer Start Time	FmSmMapBuffer	Proxy between CMS:Schedule Controller and CMS: Spacecraft Model.	CMS: Spacecraft Model	CMS: Schedule Controller	1/day
	FmMsATCBufferInfo	Contains a list of DAS IDs and start time of ATC buffer			
Map ATC	FmSmMapBuffer	Proxy Between CMS:Schedule Controller and CMS: Spacecraft Model.	CMS: Spacecraft Model	CMS: Schedule Controller	1/day
	FmMsLoadData	Contains list of directives for one load.			
	FmMsATCMap Request	Request to Map DAS command list into ATC buffer model and return a list of loads.			
Delete Buffers	FmSmMapBuffer	Proxy Between CMS: Load Catalog and CMS: Spacecraft Model.	CMS: Spacecraft Model	CMS: Load Catalog	1/week
	FmMsDeleteATCBuffers	Request to delete buffers.			
Update Buffer	FmSmMapBuffer	Proxy Between CMS: Load Catalog and CMS: Spacecraft Model.	CMS: Spacecraft Model	CMS: Load Catalog	5/day
	FmMsUpdateBuffer	Request to update Buffer from working to predicted and then to actual.			
Generate Compare Report	FmMsGenerate Map	Proxy between FUI and Spacecraft Model	CMS: Spacecraft Model	FUI: Control Window, Ground Script Controller, or Procedure Controller	1/month

Table 3.5.2. Spacecraft Model Interfaces (2 of 3)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
	FoMsCompare Request	Request to compare two image files with specified mask and start address.			
	FoMsCMSStatu s	Status of compare request.			
Generate Image Report	FmMsGenerate Map	Proxy between FUI and Spacecraft Model	CMS: Spacecraft Model	FUI: Report Generator	1/month
	FoMsImageRpt Req	Request to publish a report on a given image file.			
	FoMsCMSStatu s	Status of report request.			
Generate Map Report	FmMsGenerate Map	Proxy between FUI and Spacecraft Model	CMS: Spacecraft Model	FUI: Report Generator	1/week
	FoMsGenMapR equest	Publish a report on the given buffer locations (RTS or ATC).			
	FoMsCMSStatu s	Status of report request.			
Build Table Load Contents from Dump	FmMsGenerate Map	Proxy between FUI and Spacecraft Model	CMS: Spacecraft Model	FUI: Table Builder	5/day
	FoMsTableData Req	Request to import dump image for table load contents building			
	FoLiLoadConte nts	Generated Table load contents			
Overwrite Ground Image	FmMsGenerate Map	Proxy between FUI and Spacecraft Model	CMS: Spacecraft Model	FUI: Control Window	5/day
	FoMsImageOve rWrite	Dump image and location to overwrite			

__Table 3.5.2. Spacecraft Model Interfaces (3 of 3)__

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
	FoMsCMSStatus	Status of overwrite request.			
Provide Buffer Information	FmMsGenerateMap	Proxy between FUI and Spacecraft Model	CMS: Spacecraft Model	FUI: ATCBuffer Display or RTSBuffer Display	1/week
	FmMsBufferRequest	Request for buffer information			
Provide Buffer List	FmMsGenerateMap	Proxy between FUI and Spacecraft Model	CMS: Spacecraft Model	FUI: ATCBuffer Display or RTSBuffer Display	1/week
	FmMsBufferListRequest	Request for all buffers of specified type (ATC, RTS).			
Retrieve ATC Buffer	FmMsGenerateMap	Proxy between FUI and Spacecraft Model. Request for named ATC Buffer.	CMS: Spacecraft Model	FUI: ATCBuffer Display	5/day
Archive Buffer Model	FoFdArchive	Proxy between DMS and CMS: Spacecraft requesting archiving of the buffer model	DMS: File Manager	CMS: Spacecraft	1/day
Send Events	FoFdEventLogger	Proxy between DMS and CMS: Spacecraft requesting broadcast of a message	DMS: Event Logger	CMS: Spacecraft	10/day
Get Table Format	FoFdGetTableFormat	Proxy between DMS and CMS: Spacecraft requesting a table format	DMS: Sybase	CMS: Spacecraft	1/day

3.5.3 Spacecraft Model Object Model

FmSmSpacecraftModel manages the spacecraft buffer modeling (see Figure 3.5-2 through 3.5-7). It manages the ATC buffer model, the RTS buffer models, the Table buffer models and the ground images. FmSmSpacecraftModel receives the FmMsATCMapRequest from the CMS Schedule Controller. When an FmMsATCMapRequest is received FmSmSpacecraftModel creates an FmSmATCBufferModel object from the most recent predicted FmSmATCBufferModel. In the event of a late change, FmSmSpacecraftModel will delete the predicted FmSmATCBufferModels from the available list. FmSmSpacecraftModel is responsible for archiving expired FmSmATCBufferModels with DMS, and retrieves the buffers upon a FUI request. The FmSmRTSBufferModel and FmSmTableModel are updated when an FmMsUpdateBuffer. At that point the FmSmRTSBufferModel affected will map the command list into the FmSmRTSBufferModel; it is also at this point that the FmSmGroundImage is updated. The RTS FoLiLoadImage is written into the ground image, FmSmRTSImage. FmSmTableModel updates are handled similarly to the FmSmRTSBufferModel. When an FmMsUpdateBuffer is received the FmSmTables are updated with the correct data values and the FmSmGroundImage, FmSmTableImage is updated with the FoLiLoadImage.

FmSmATCBufferModel determines the number of command locations available in the buffer, determines the uplink window for the load, determines if a DAS needs to be partitioned into more than one load, keeps activities from being split, adds safe commands to the end of the load, updates the FmSmATCBufferModel status from working to predicted to actual to previous, and generates an FoRpMapReport upon request. For each partitioned load, FmSmATCBufferModel returns FmMsLoadData.

FmSmRTSBufferModel manages all of the FmSmRTSBuffers. It requests the particular FmSmRTSBuffer to produce its FoRpMapReport, it retrieves the requested FmSmRTSBuffer, and requests the FmSmRTSBuffer to update itself.

FmSmRTSBuffer represents a single RTS buffer. It generates its FoRpMapReport and updates the FmSmRTSBuffer with the RTS FoLiLoadContents.

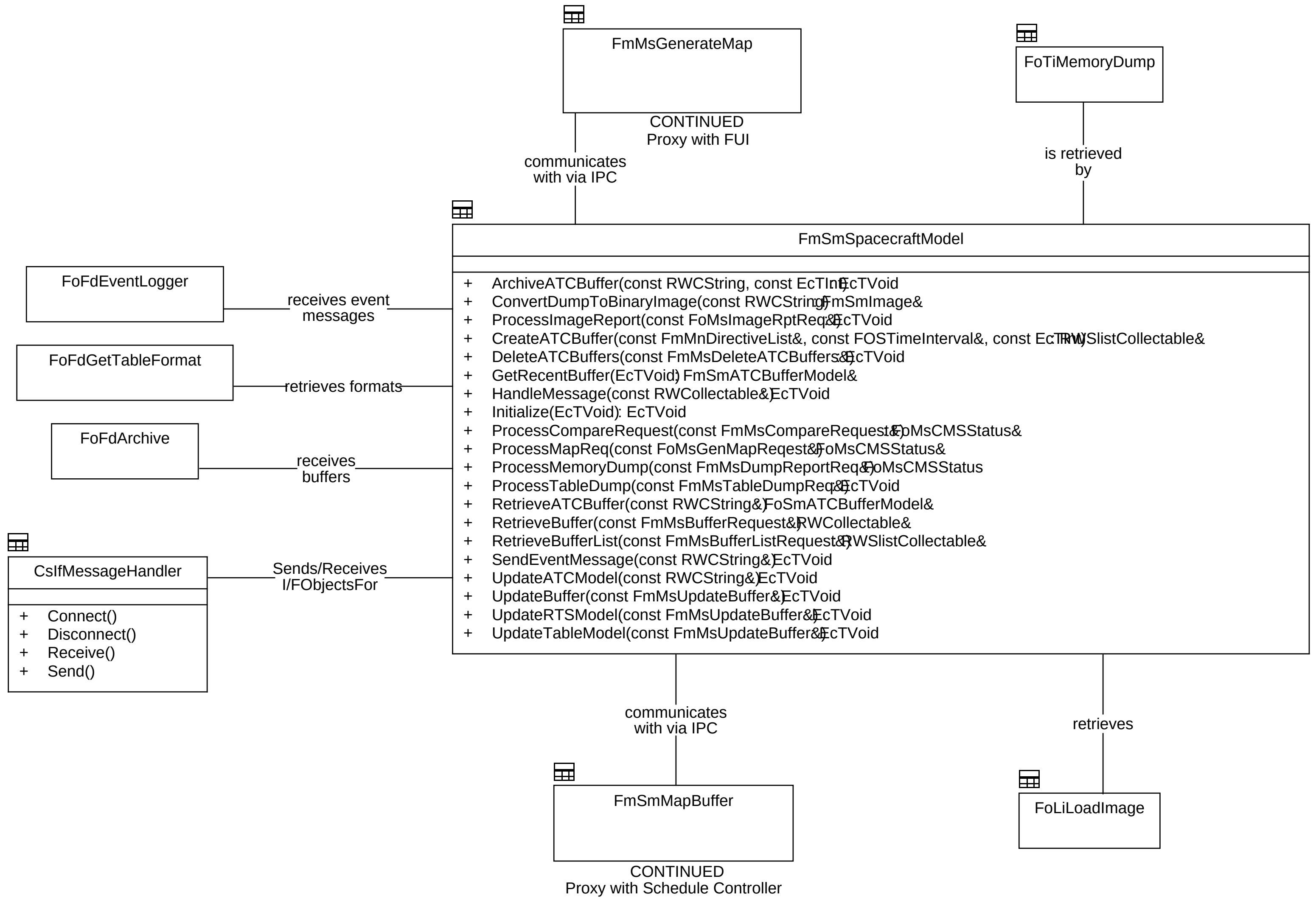

FmMsGenerateMap
CONTINUED
Proxy with FUI
FoTiMemoryDump
communicates with via IPC
is retrieved by
FmSmSpacecraftModel
+ ArchiveATCBuffer(const RWCString, const EcTInt): EcTVoid
+ ConvertDumpToBinaryImage(const RWCString): FmSmImage&
+ ProcessImageReport(const FoMsImageRptReq&): EcTVoid
+ CreateATCBuffer(const FmMnDirectiveList&, const FOSTimeInterval&, const EcTInt): RWSlistCollectable&
+ DeleteATCBuffers(const FmMsDeleteATCBuffers&): EcTVoid
+ GetRecentBuffer(EcTVoid): FmSmATCBufferModel&
+ HandleMessage(const RWCollectable&): EcTVoid
+ Initialize(EcTVoid): EcTVoid
+ ProcessCompareRequest(const FmMsCompareRequest&): FoMsCMSStatus&
+ ProcessMapReq(const FoMsGenMapReqest&): FoMsCMSStatus&
+ ProcessMemoryDump(const FmMsDumpReportReq&): FoMsCMSStatus
+ ProcessTableDump(const FmMsTableDumpReq&): EcTVoid
+ RetrieveATCBuffer(const RWCString&): FoSmATCBufferModel&
+ RetrieveBuffer(const FmMsBufferRequest&): RWCollectable&
+ RetrieveBufferList(const FmMsBufferListRequest&): RWSlistCollectable&
+ SendEventMessage(const RWCString&): EcTVoid
+ UpdateATCModel(const RWCString&): EcTVoid
+ UpdateBuffer(const FmMsUpdateBuffer&): EcTVoid
+ UpdateRTSModel(const FmMsUpdateBuffer&): EcTVoid
+ UpdateTableModel(const FmMsUpdateBuffer&): EcTVoid
FoFdEventLogger
receives event messages
FoFdGetTableFormat
retrieves formats
FoFdArchive
receives buffers
CsIfMessageHandler
+ Connect()
+ Disconnect()
+ Receive()
+ Send()
Sends/Receives I/FObjectsFor
communicates with via IPC
retrieves
FmSmMapBuffer
CONTINUED
Proxy with Schedule Controller
FoLiLoadImage

Figure 3.5-2. Spacecraft Model Object Model (1 of 6)

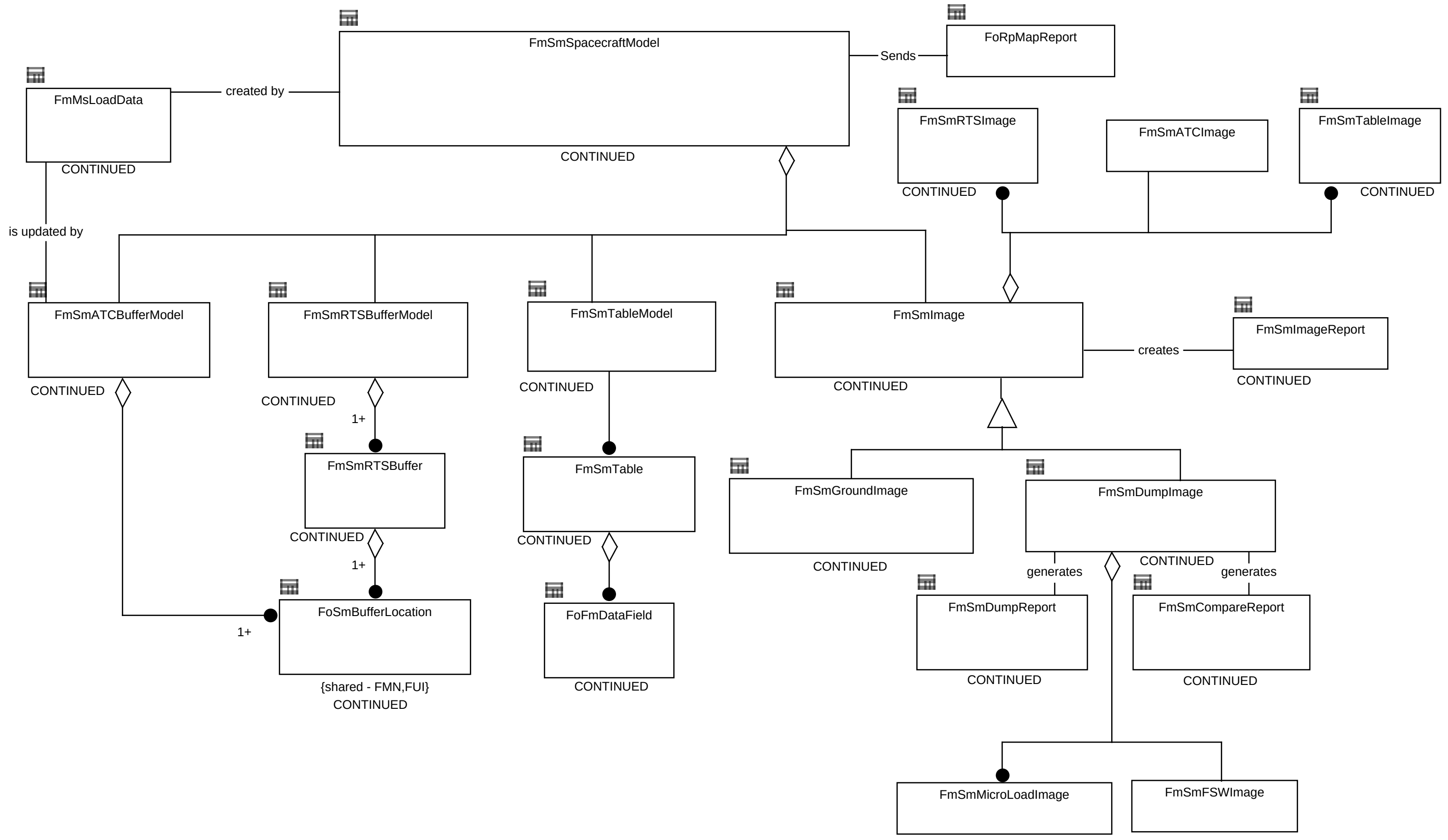


Figure 3.5-3. Spacecraft Model Object Model (2 of 6)

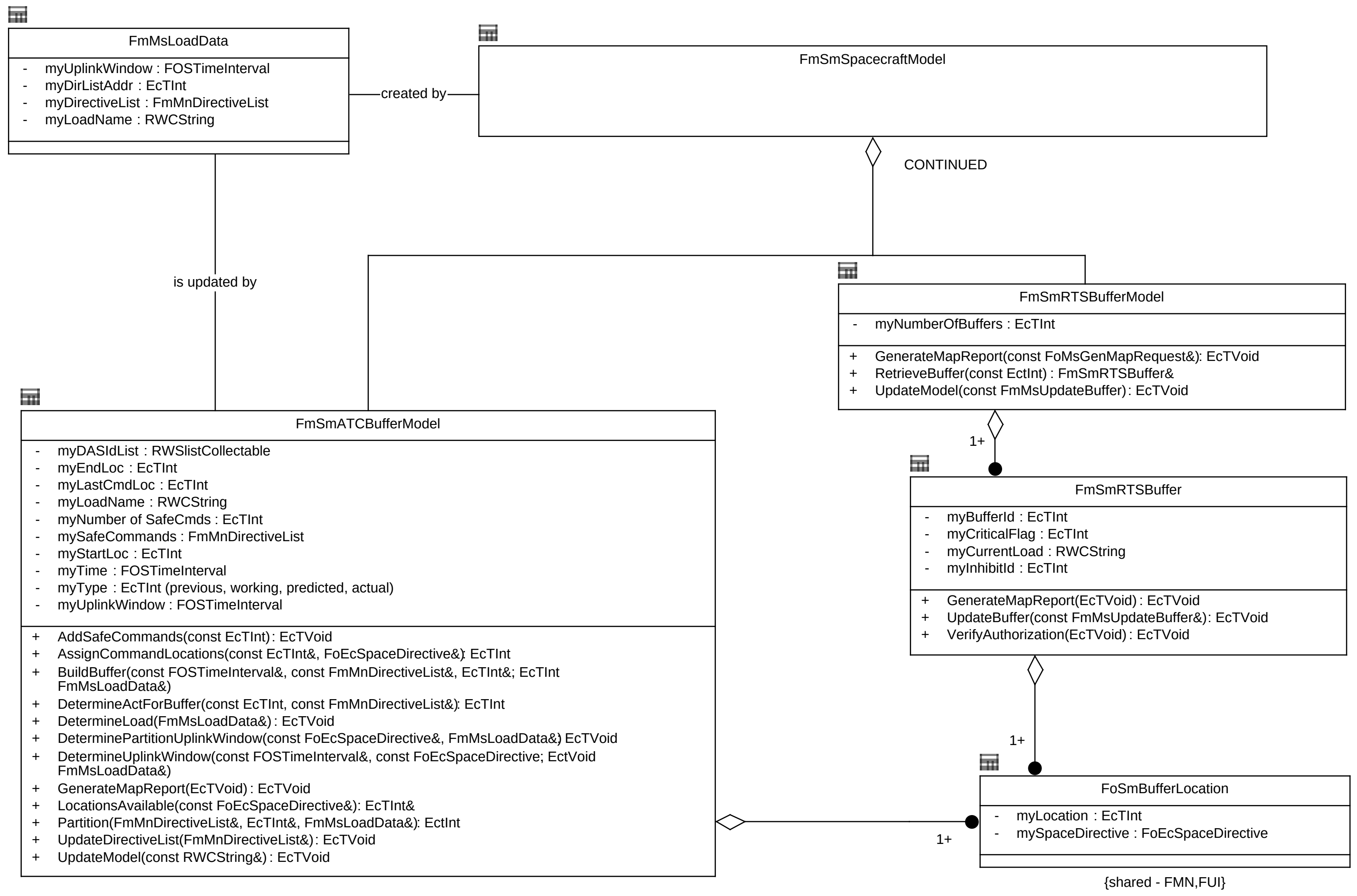

FmMsLoadData
- myUplinkWindow : FOSTimeInterval
- myDirListAddr : EcTInt
- myDirectiveList : FmMnDirectiveList
- myLoadName : RWCString
created by
FmSmSpacecraftModel
CONTINUED
is updated by
FmSmRTSBufferModel
- myNumberOfBuffers : EcTInt
+ GenerateMapReport(const FoMsGenMapRequest&): EcTVoid
+ RetrieveBuffer(const EctInt) : FmSmRTSBuffer&
+ UpdateModel(const FmMsUpdateBuffer) : EcTVoid
FmSmATCBufferModel
- myDASIdList : RWSlistCollectable
- myEndLoc : EcTInt
- myLastCmdLoc : EcTInt
- myLoadName : RWCString
- myNumber of SafeCmds : EcTInt
- mySafeCommands : FmMnDirectiveList
- myStartLoc : EcTInt
- myTime : FOSTimeInterval
- myType : EcTInt (previous, working, predicted, actual)
- myUplinkWindow : FOSTimeInterval
+ AddSafeCommands(const EcTInt) : EcTVoid
+ AssignCommandLocations(const EcTInt&, FoEcSpaceDirective&): EcTInt
+ BuildBuffer(const FOSTimeInterval&, const FmMnDirectiveList&, EcTInt&; EcTInt FmMsLoadData&)
+ DetermineActForBuffer(const EcTInt, const FmMnDirectiveList&): EcTInt
+ DetermineLoad(FmMsLoadData&) : EcTVoid
+ DeterminePartitionUplinkWindow(const FoEcSpaceDirective&, FmMsLoadData&) EcTVoid
+ DetermineUplinkWindow(const FOSTimeInterval&, const FoEcSpaceDirective; EctVoid FmMsLoadData&)
+ GenerateMapReport(EcTVoid) : EcTVoid
+ LocationsAvailable(const FoEcSpaceDirective&): EcTInt&
+ Partition(FmMnDirectiveList&, EcTInt&, FmMsLoadData&): EctInt
+ UpdateDirectiveList(FmMnDirectiveList&) : EcTVoid
+ UpdateModel(const RWCString&) : EcTVoid
1+
FmSmRTSBuffer
- myBufferId : EcTInt
- myCriticalFlag : EcTInt
- myCurrentLoad : RWCString
- myInhibitId : EcTInt
+ GenerateMapReport(EcTVoid) : EcTVoid
+ UpdateBuffer(const FmMsUpdateBuffer&): EcTVoid
+ VerifyAuthorization(EcTVoid) : EcTVoid
1+
FoSmBufferLocation
- myLocation : EcTInt
- mySpaceDirective : FoEcSpaceDirective
1+
{shared - FMN,FUI}

Figure 3.5-4. Spacecraft Model Object Model (3 of 6)

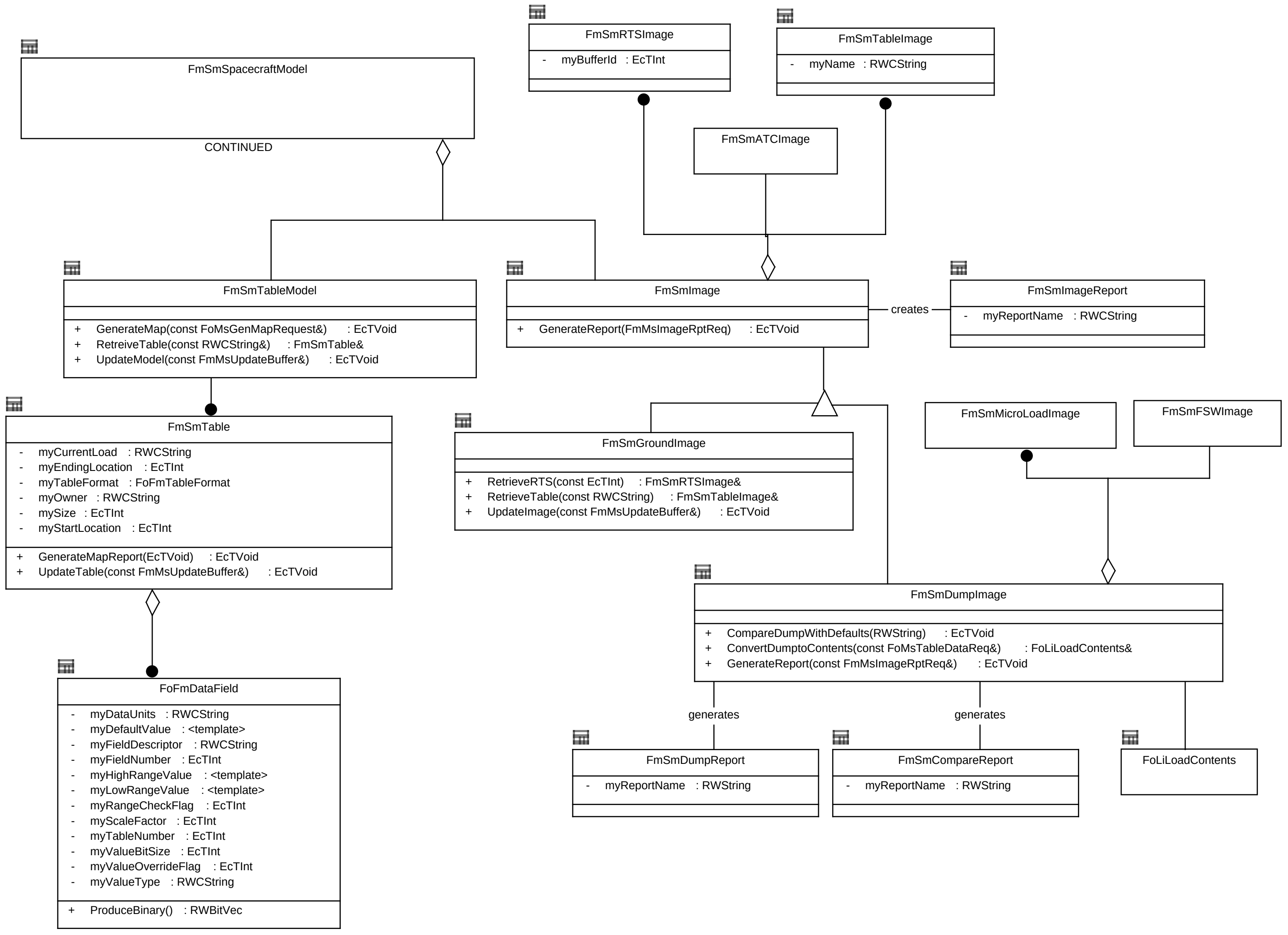

FmSmSpacecraftModel
CONTINUED
FmSmRTSImage
- myBufferId : EcTInt
FmSmTableImage
- myName : RWCString
FmSmATCImage
FmSmTableModel
+ GenerateMap(const FoMsGenMapRequest&) : EcTVoid
+ RetreiveTable(const RWCString&) : FmSmTable&
+ UpdateModel(const FmMsUpdateBuffer&) : EcTVoid
FmSmImage
+ GenerateReport(FmMsImageRptReq) : EcTVoid
creates
FmSmImageReport
- myReportName : RWCString
FmSmTable
- myCurrentLoad : RWCString
- myEndingLocation : EcTInt
- myTableFormat : FoFmTableFormat
- myOwner : RWCString
- mySize : EcTInt
- myStartLocation : EcTInt
+ GenerateMapReport(EcTVoid) : EcTVoid
+ UpdateTable(const FmMsUpdateBuffer&) : EcTVoid
FmSmGroundImage
+ RetrieveRTS(const EcTInt) : FmSmRTSImage&
+ RetrieveTable(const RWCString) : FmSmTableImage&
+ UpdateImage(const FmMsUpdateBuffer&) : EcTVoid
FmSmMicroLoadImage
FmSmFSWImage
FmSmDumpImage
+ CompareDumpWithDefaults(RWString) : EcTVoid
+ ConvertDumptoContents(const FoMsTableDataReq&) : FoLiLoadContents&
+ GenerateReport(const FmMsImageRptReq&) : EcTVoid
generates
generates
FoFmDataField
- myDataUnits : RWCString
- myDefaultValue : <template>
- myFieldDescriptor : RWCString
- myFieldNumber : EcTInt
- myHighRangeValue : <template>
- myLowRangeValue : <template>
- myRangeCheckFlag : EcTInt
- myScaleFactor : EcTInt
- myTableNumber : EcTInt
- myValueBitSize : EcTInt
- myValueOverrideFlag : EcTInt
- myValueType : RWCString
+ ProduceBinary() : RWBitVec
FmSmDumpReport
- myReportName : RWString
FmSmCompareReport
- myReportName : RWString
FoLiLoadContents

Figure 3.5-5. Spacecraft Model Object Model (4 of 6)

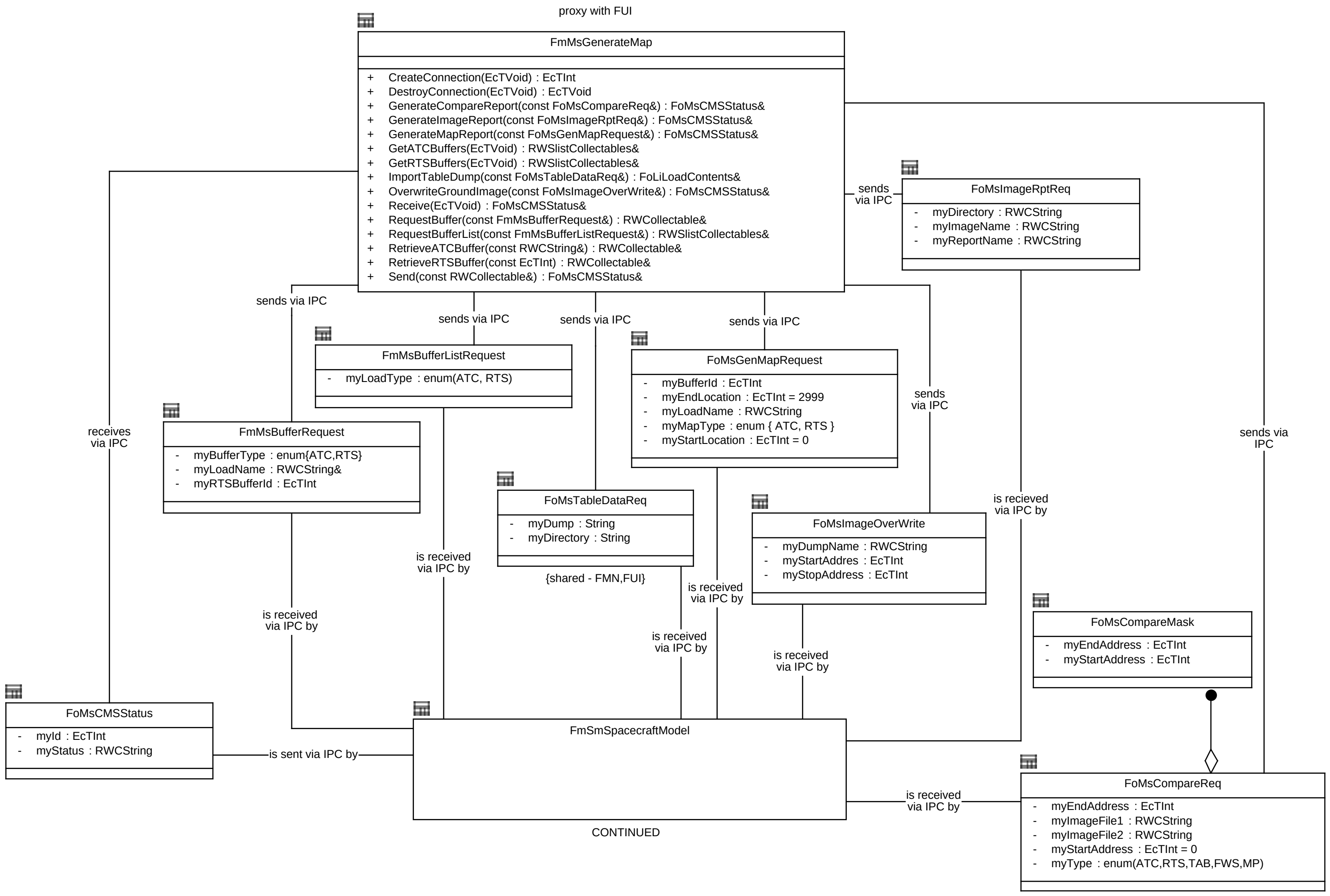

proxy with FUI
FmMsGenerateMap
+ CreateConnection(EcTVoid) : EcTInt
+ DestroyConnection(EcTVoid) : EcTVoid
+ GenerateCompareReport(const FoMsCompareReq&) : FoMsCMSStatus&
+ GenerateImageReport(const FoMsImageRptReq&) : FoMsCMSStatus&
+ GenerateMapReport(const FoMsGenMapRequest&) : FoMsCMSStatus&
+ GetATCBuffers(EcTVoid) : RWSlistCollectables&
+ GetRTSBuffers(EcTVoid) : RWSlistCollectables&
+ ImportTableDump(const FoMsTableDataReq&) : FoLiLoadContents&
+ OverwriteGroundImage(const FoMsImageOverWrite&) : FoMsCMSStatus&
+ Receive(EcTVoid) : FoMsCMSStatus&
+ RequestBuffer(const FmMsBufferRequest&) : RWCollectable&
+ RequestBufferList(const FmMsBufferListRequest&) : RWSlistCollectables&
+ RetrieveATCBuffer(const RWCString&) : RWCollectable&
+ RetrieveRTSBuffer(const EcTInt) : RWCollectable&
+ Send(const RWCollectable&) : FoMsCMSStatus&
sends via IPC
FoMsImageRptReq
- myDirectory : RWCString
- myImageName : RWCString
- myReportName : RWCString
sends via IPC
sends via IPC
sends via IPC
sends via IPC
FmMsBufferListRequest
- myLoadType : enum(ATC, RTS)
FoMsGenMapRequest
- myBufferId : EcTInt
- myEndLocation : EcTInt = 2999
- myLoadName : RWCString
- myMapType : enum { ATC, RTS }
- myStartLocation : EcTInt = 0
sends via IPC
sends via IPC
receives via IPC
FmMsBufferRequest
- myBufferType : enum{ATC,RTS}
- myLoadName : RWCString&
- myRTSBufferId : EcTInt
FoMsTableDataReq
- myDump : String
- myDirectory : String
{shared - FMN,FUI}
FoMsImageOverWrite
- myDumpName : RWCString
- myStartAddres : EcTInt
- myStopAddress : EcTInt
is recieved via IPC by
is received via IPC by
is received via IPC by
is received via IPC by
is received via IPC by
is received via IPC by
FoMsCompareMask
- myEndAddress : EcTInt
- myStartAddress : EcTInt
FoMsCMSStatus
- myId : EcTInt
- myStatus : RWCString
is sent via IPC by
FmSmSpacecraftModel
CONTINUED
is received via IPC by
FoMsCompareReq
- myEndAddress : EcTInt
- myImageFile1 : RWCString
- myImageFile2 : RWCString
- myStartAddress : EcTInt = 0
- myType : enum(ATC,RTS,TAB,FWS,MP)

Figure 3.5-6. Spacecraft Model Object Model (5 of 6)

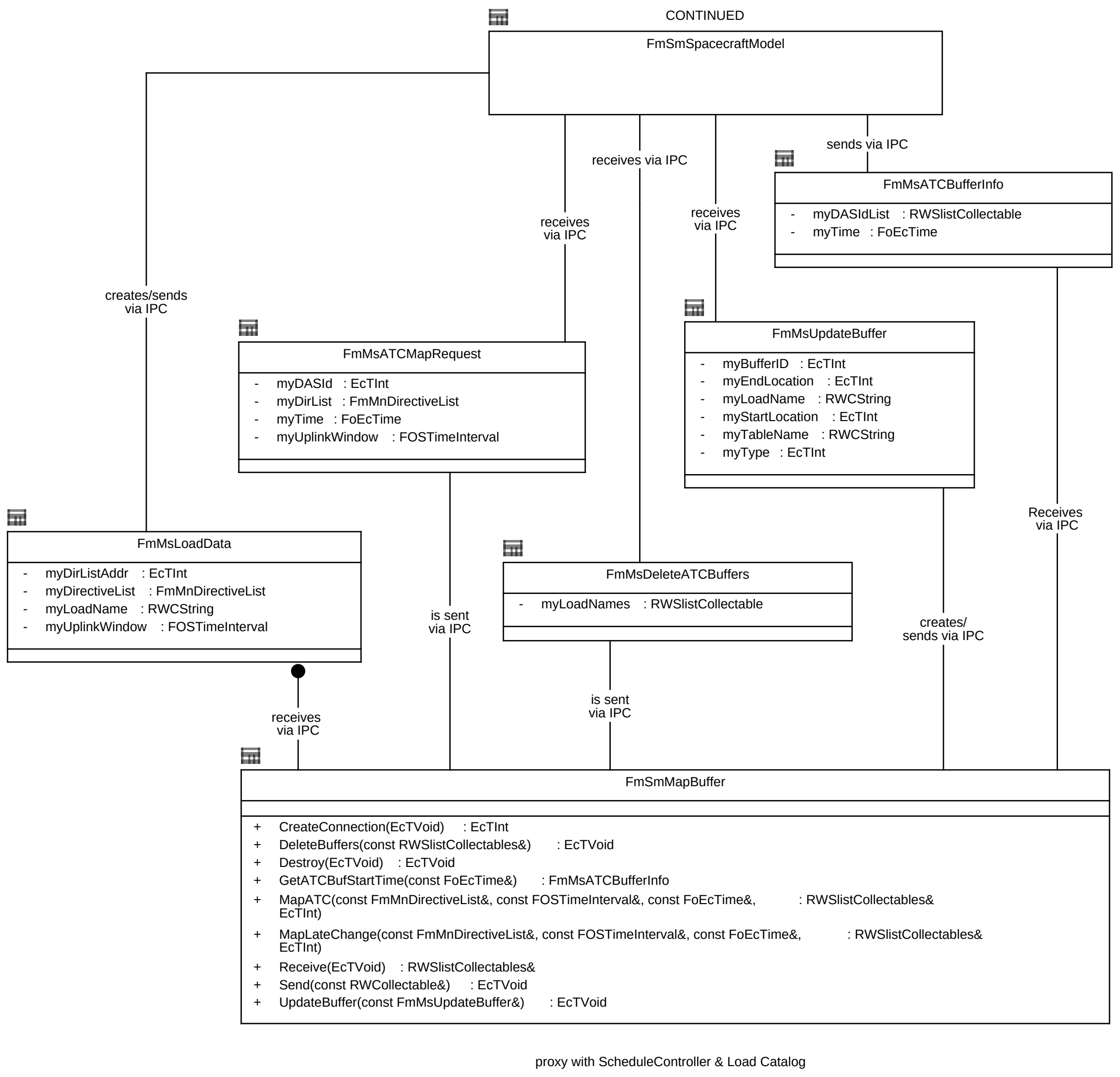

CONTINUED
FmSmSpacecraftModel
creates/sends via IPC
receives via IPC
receives via IPC
receives via IPC
sends via IPC
FmMsATCBufferInfo
- myDASIdList : RWSlistCollectable
- myTime : FoEcTime
FmMsUpdateBuffer
- myBufferID : EcTInt
- myEndLocation : EcTInt
- myLoadName : RWCString
- myStartLocation : EcTInt
- myTableName : RWCString
- myType : EcTInt
FmMsATCMapRequest
- myDASId : EcTInt
- myDirList : FmMnDirectiveList
- myTime : FoEcTime
- myUplinkWindow : FOSTimeInterval
FmMsLoadData
- myDirListAddr : EcTInt
- myDirectiveList : FmMnDirectiveList
- myLoadName : RWCString
- myUplinkWindow : FOSTimeInterval
FmMsDeleteATCBuffers
- myLoadNames : RWSlistCollectable
Receives via IPC
is sent via IPC
creates/ sends via IPC
is sent via IPC
receives via IPC
FmSmMapBuffer
+ CreateConnection(EcTVoid) : EcTInt
+ DeleteBuffers(const RWSlistCollectables&) : EcTVoid
+ Destroy(EcTVoid) : EcTVoid
+ GetATCBufStartTime(const FoEcTime&) : FmMsATCBufferInfo
+ MapATC(const FmMnDirectiveList&, const FOSTimeInterval&, const FoEcTime&, EcTInt) : RWSlistCollectables&
+ MapLateChange(const FmMnDirectiveList&, const FOSTimeInterval&, const FoEcTime&, EcTInt) : RWSlistCollectables&
+ Receive(EcTVoid) : RWSlistCollectables&
+ Send(const RWCollectable&) : EcTVoid
+ UpdateBuffer(const FmMsUpdateBuffer&) : EcTVoid
proxy with ScheduleController & Load Catalog

Figure 3.5-7. Spacecraft Model Object Model (6 of 6)

FoSmBufferLocation represents a single buffer location for either an FmSmRTSBuffer or an FmSmATCBufferModel.

FmSmTableModel manages all of the FmSmTables. It retrieves a particular FmSmTable, requests FmSmTable to generate its FoRpMapReport and requests the FmSmTable to update its model.

FmSmTable represents a single table. It generates its FoRpMapReport and updates itself with the with the Table FoLiLoadContents.

FoFmDataField represents a single data field in a FmSmTable.

FmSmImage represents an abstract class for the FmSmGroundImage and the FmSmDumpImage. A FmSmGroundImage consists of an FmSmRTSImage, an FmSmATCImage, and a FmSmTableImage. A FmSmDumpImage consists of these three and a FmSmMicroLoadImage and a FmSmFSWImage.

FmSmGroundImage represents the following spacecraft images: RTS, ATC, and Table. It is responsible for updating the respective image when an FmMsUpdateBuffer is received.

FmSmDumpImage is responsible for comparing the values in a table dump to their corresponding default values in DMS, generating a FmSmDumpReport on a specified dump file and for converting a specified table dump into a FoLiLoadContents. FmSmDumpImage receives FoMsImageRptReq when generating a FmSmDumpReport and FoMsTableDataReq when converting a dump to a FoLiLoadContents.

FmSmRTSImage represents an image of a single RTS buffer. This class is used for both the ground images and the dump images

FmSmATCImage represents an image of the ATC buffer. This class is used for both ground and dump images

FmSmTableImage represents an image of a single table. This class is used for both ground and dump images.

FmSmMicroLoadImage represents a dump image for a single microprocessor.

FmSmFSWImage represents a dump image for a single flight software image.

FmSmDumpReport represents the report generated from an FoMsImageRptReq.

FmSmCompareReport represents the report generated from an FoMsCompareReq.

FmMsLoadData includes the information for generating an ATC load. It has the directive list from the DAS that will fit into the buffer, it includes the uplink window, the load name and the address of the next command to be included in a partitioned load.

FmMsATCMapRequest includes information for processing the DAS command list. It includes the DAS Id, the DAS directive list, the time of the first command to be included in the buffer, and the requested uplink window.

FmMsDeleteATCBuffers is the message received by FmSmSpacecraftModel to delete the predicted FmSmATCBufferModels.

FmMsUpdateBuffer is sent to FmSmSpacecraftModel to update the status of the FmSmATCBufferModel, the FmSmRTSBufferModel, the FmSmTableModel. It will update the FmSmATCBufferModel from a working status buffer to a predicted status buffer and update a predicted status buffer to the actual FmSmATCBufferModel. The FmSmRTSBufferModel and the

FmSmTableModel are updated from the FoLiLoadContents. The FmSmGroundImages for the FmSmATCBufferModel, the FmSmRTSBuffers and the FmSmTables are also updated from this FmMsUpdateBuffer request.

FmMsATCBufferInfo is sent by FmSmSpacecraftModel in response to a request from CMS Schedule Controller. It returns information on the most recent predicted FmSmATCBufferModel. The information returned, the DAS id list of a buffer, and the time of the first command in the buffer is used for constraint checking the DAS command list about to be processed.

FmSmMapBuffer is a proxy class used internally by CMS. Both CMS Schedule Controller and CMS Load Catalog incorporate this class. It performs the interprocess communication between these processes and FmSmSpacecraftModel, allowing the buffers to be created, updated, and deleted. FmSmMapBuffer creates FmMsATCMapRequest, FmMsDeleteATCBuffers, and FmMsUpdateBuffer. It receives FmMsLoadData and FmMsATCBufferInfo.

FmMsGenerateMap is a proxy class used by FUI to request information from FmSmSpacecraftModel model for display and reporting information. It creates FmMsBufferRequest and FmMsBufferListRequest and receives the following objects: FoMsTableDataReq, FoMsImageRptReq, FoMsImageOverWrite, FoMsCompareReq and FoMsGenMapRequest which are sent to FmSmSpacecraftModel. It receives FoMsCMSStatus from FmSmSpacecraftModel.

FmMsBufferRequest is the class that requests a specific FmSmATCBufferModel or FmSmRTSBuffer from FmSmSpacecraftModel.

FmMsBufferListRequest is a request to FmSmSpacecraftModel to return the list of available FmSmRTSBuffers or FmSmATCBufferModels.

FoMsGenMapRequest is a request to FmSmSpacecraftModel to generate a FoRpMapReport for the specified buffer, either FmSmATCBufferModel, FmSmRTSBuffer(s), or FmSmTable(s).

FoMsTableDataReq is sent from FUI. It is a request from the table load builder to import a table FoTlMemoryDump and convert it into a table FoLiLoadContents.

FoMsImageOverWrite is sent from FUI. It is a request to overwrite a portion of the FmSmGroundImage with either a FoLiLoadImage or a FmSmDumpImage.

FoMsImageRptReq is sent from FUI. It is a request to generate an FmSmImageReport.

FoMsCompareReq is sent from FUI. It is a request to generate a FmSmCompareReport. The user can compare any two image files.

FoMsCompareMask is part of FoMsCompareReq. It allows the user to select a portion of the image files not to compare.

3.5.4 Spacecraft Model Dynamic Model

The Spacecraft Model had the following scenarios:

- Initialization
- ATC Load Generation
- ATC Buffer Model Update
- ATC Buffer Model Deletion
- ATC Buffer Display Request
- RTS Buffer Display Request
- Map Report Generation
- Image Report Generation
- Compare Report Generation
- Table Model & Image Update
- RTS Buffer Model & Image Update
- ATC Buffer Model & Image Update
- Flight Software Image Update
- Table Data Request
- Ground Image Overwrite

3.5.4.1 Initialization Scenario

3.5.4.1.1 Initialization Scenario Abstract

This scenario occurs when the Spacecraft Model process is started (see Figure 3.5-8). It addresses the initialization of the Spacecraft interfaces and loading of configuration files.

3.5.4.1.2 Initialization Summary Information

Interfaces:

- DMS

Stimulus:

- Spacecraft Model process is started

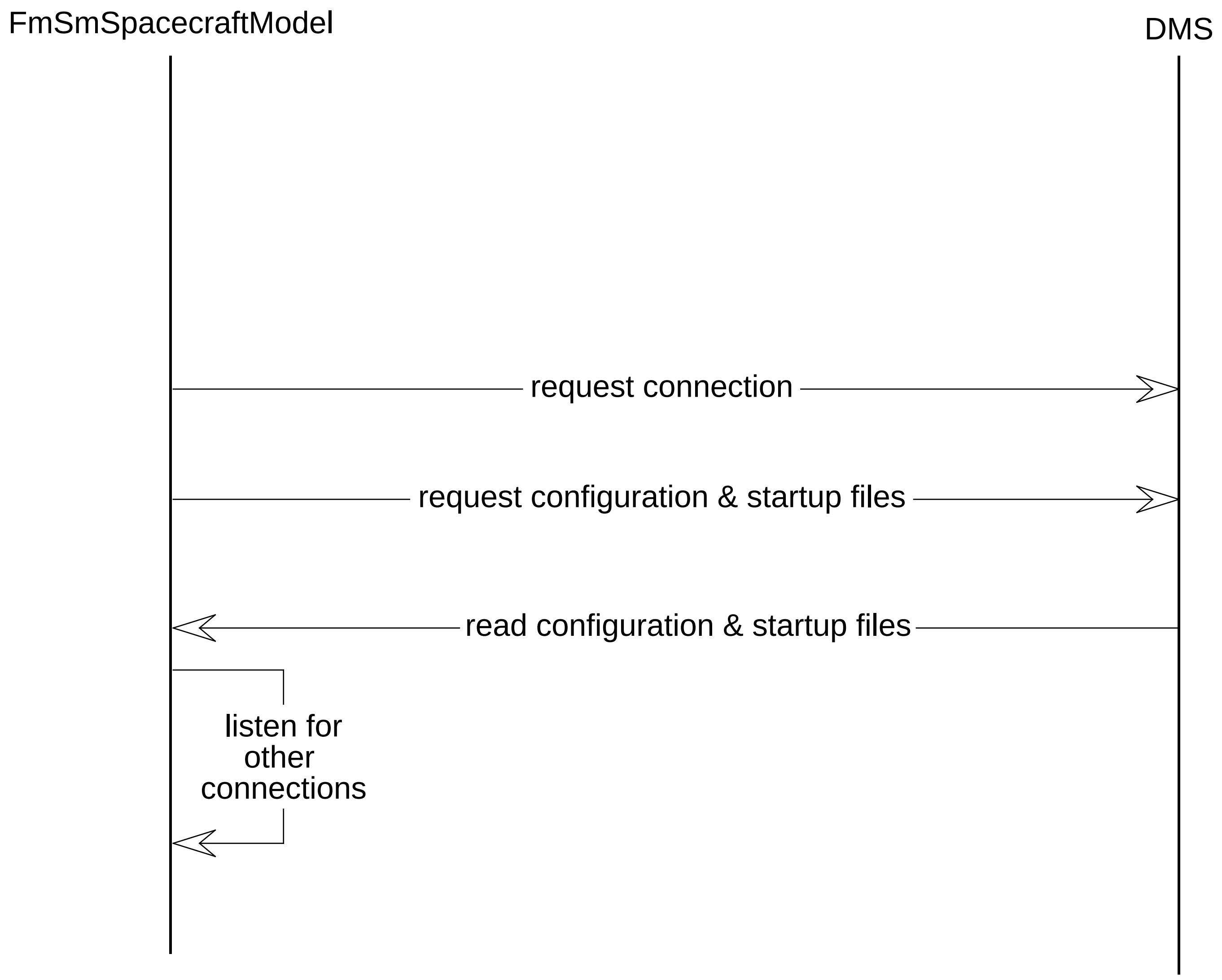

FmSmSpacecraftModel
DMS
request connection
request configuration & startup files
read configuration & startup files
listen for other connections

Figure 3.5-8 . Spacecraft Model Initialization Event Trace

Desired Response:

- Spacecraft Model is up and running

Pre-Conditions:

- DMS software has been initiated

Post-Conditions:

- Spacecraft Model is ready to process requests

3.5.4.1.3 Initialization Scenario Description

When the Spacecraft Model is started FmSmSpacecraftModel will initialize its interfaces by requesting address information from the Name Server. Once the interface connections have been made FmSmSpacecraftModel will read in the current list of ATC buffers, RTS buffers, tables, ground images and it will read in the "safe" commands from DMS.

3.5.4.2 ATC Load Generation Scenario

3.5.4.2.1 ATC Load Generation Scenario Abstract

The ATC Load Generation scenario describes the receipt and processing of an Expanded Directive List from CMS Schedule Controller (FmScScheduleController) via FmMsValidateConstraints proxy (see Figure 3.5-9).

3.5.4.2.2 ATC Load Generation Summary Information

Interfaces:

- CMS Schedule Controller

Stimulus:

- Receipt of DAS Expanded Directive List

Desired Response:

- ATC commands are mapped into the buffer

Pre-Conditions:

- Schedule controller software has been initiated
- Spacecraft model software has been initiated

Post-Conditions:

- Working ATC Buffer Model is created.

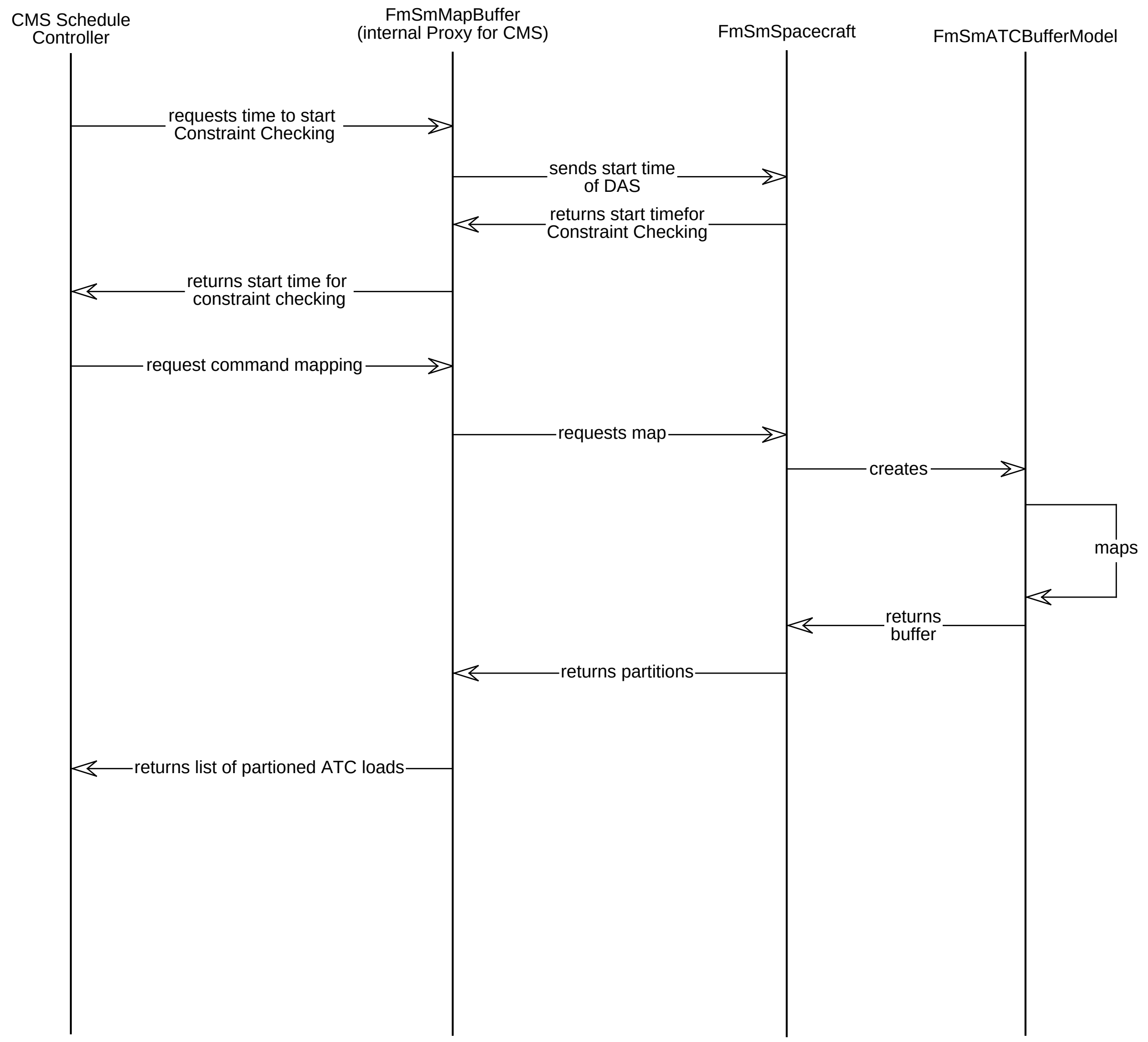

CMS Schedule Controller
FmSmMapBuffer (internal Proxy for CMS)
FmSmSpacecraft
FmSmATCBufferModel
requests time to start Constraint Checking
sends start time of DAS
returns start timefor Constraint Checking
returns start time for constraint checking
request command mapping
requests map
creates
maps
returns buffer
returns partitions
returns list of partioned ATC loads

Figure 3.5-9. Spacecraft Model ATC Load Generation Event Trace

3.5.4.2.3 ATC Load Generation Scenario Description

The Spacecraft Model receives a request via FmSmMapBuffer to determine the buffer that the new DAS being processed will be input. The start time of this buffer is returned to the CMS Schedule Controller so that the DAS can be constraint checked. The constraint free expanded directive list from the DAS is sent to the Spacecraft Model from the Schedule Controller. The Spacecraft Model sends the directive list to the ATC buffer, which maps the commands into the buffer and determines the uplink window. For each partition a FmMsLoadData is returned.

3.5.4.3 ATC Buffer Model Update Scenario

3.5.4.3.1 ATC Buffer Model Update Scenario Abstract

The Spacecraft Model receives an Update Buffer request from the CMS Schedule controller or CMS Load Catalog. Based on the request the ATC buffer status is updated from a working buffer to a predicted buffer or from a predicted buffer to and actual buffer. When the request is received that changes the status from predicted to actual, the current actual buffer is updated to previous status (see Figure 3.5-10).

3.5.4.3.2 ATC Buffer Model Update Summary Information

Interfaces:

- CMS Schedule
- CMS Load Catalog

Stimulus:

- Receipt of Update Buffer request

Desired Response:

- Buffer model is updated to reflect new status

Pre-Conditions:

- Schedule Controller software has been initiated
- Load Catalog software has been initiated
- Spacecraft model software has been initiated

Post-Conditions:

- ATC Buffer Models are updated

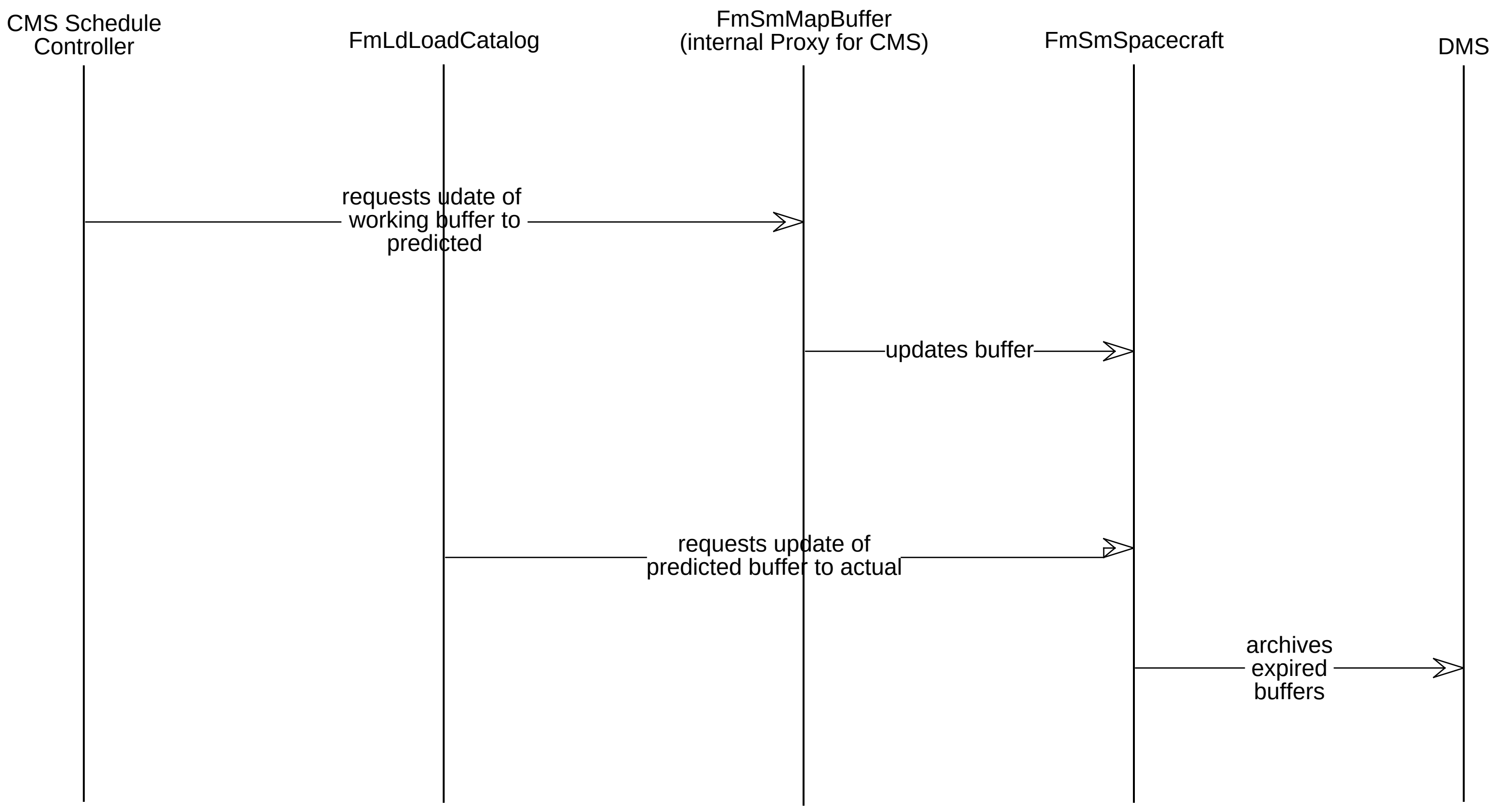

CMS Schedule Controller
FmLdLoadCatalog
FmSmMapBuffer (internal Proxy for CMS)
FmSmSpacecraft
DMS
requests udate of working buffer to predicted
updates buffer
requests update of predicted buffer to actual
archives expired buffers

Figure 3.5.10. Spacecraft Model ATC Buffer Model Update Event Trace

3.5.4.3.3 ATC Buffer Model Update Scenario Description

The ATC Buffer Model Update scenario describes the receipt of an FmSmUpdateBuffer request from either CMS Schedule Controller or CMS Load Catalog via FmSmMapBuffer proxy. The Spacecraft Model receives a request via FmSmMapBuffer.

When the new load is successfully generated, the CMS Schedule Controller requests that the working buffer be updated to predicted buffer status. When the load is successfully uplinked, the CMS Load Catalog requests that the buffer be updated to be the actual buffer model.

3.5.4.4 ATC Buffer Model Deletion Scenario

3.5.4.4.1 ATC Buffer Model Deletion Scenario Abstract

The Spacecraft Model receives the request to delete ATC buffers from CMS Load Catalog via FmSmMapBuffer proxy (see Figure 3.5-11). The request is made as a result of a late change. The predicted status ATC buffers are deleted.

3.5.4.4.2 ATC Buffer Model Deletion Summary Information

Interfaces:

- CMS Load Catalog

Stimulus:

- Receipt of Buffer Delete Request

Desired Response:

- Buffer(s) are deleted

Pre-Conditions:

- CMS Load Catalog software has been initiated
- Spacecraft model software has been initiated

Post-Conditions:

- Predicted status ATC buffer models are deleted

3.5.4.4.3 ATC Buffer Model Deletion Scenario Description

CMS Load Catalog requests ATC Buffer deletion via the FmMsMapBuffer proxy. The Spacecraft Model deletes the predicted ATC buffer models.

3.5.4.5 ATC Buffer Display Scenario

3.5.4.5.1 ATC Buffer Display Scenario Abstract

The Spacecraft Model receives the requests to provide a buffer list and to provide a specified buffer based on a selection from that list (see Figure 3.5-12). This information is used by FUI to provide the ATC buffer display for the user.

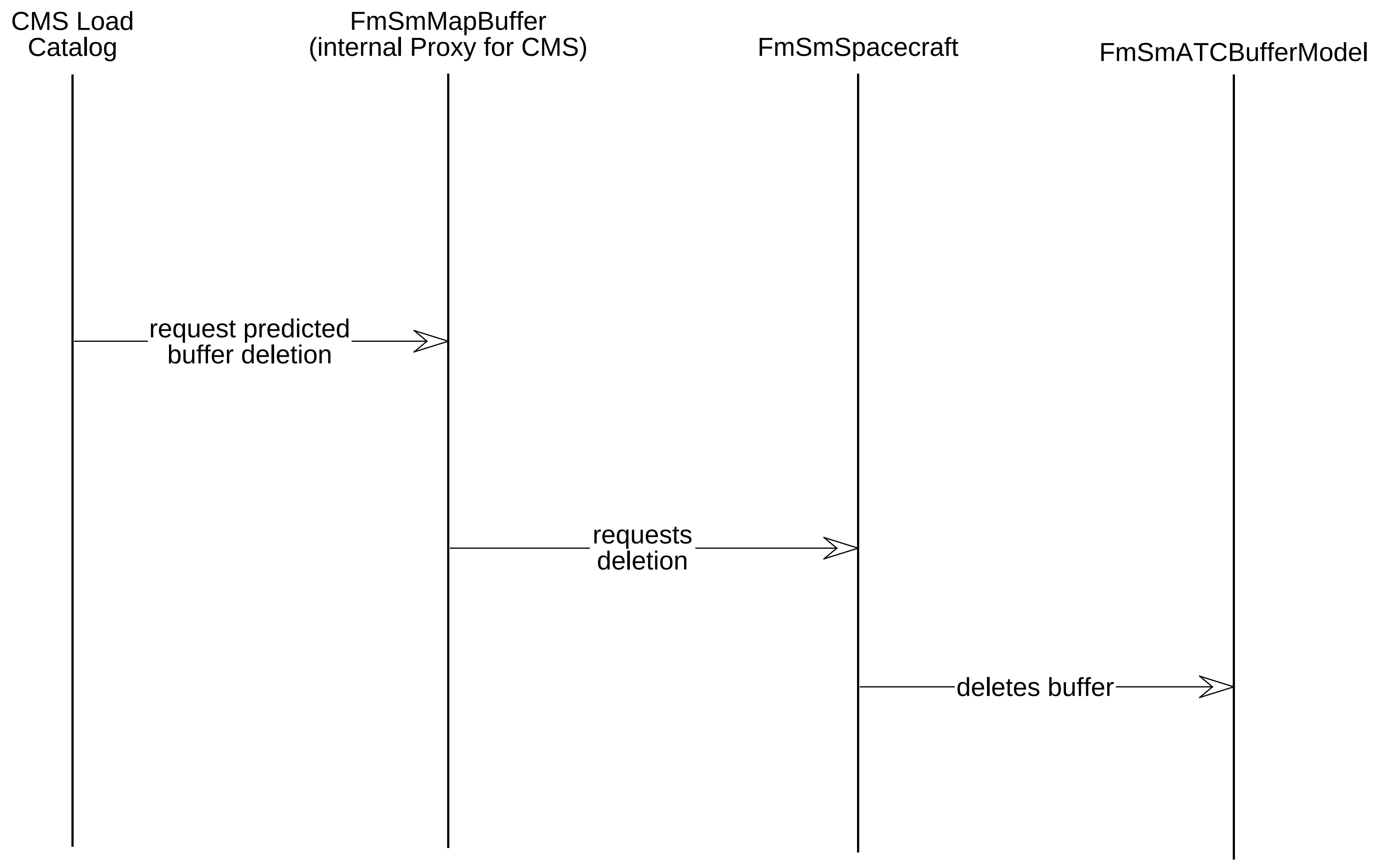


Figure 3.5-11. Spacecraft Model ATC Buffer Model Deletion Event Trace

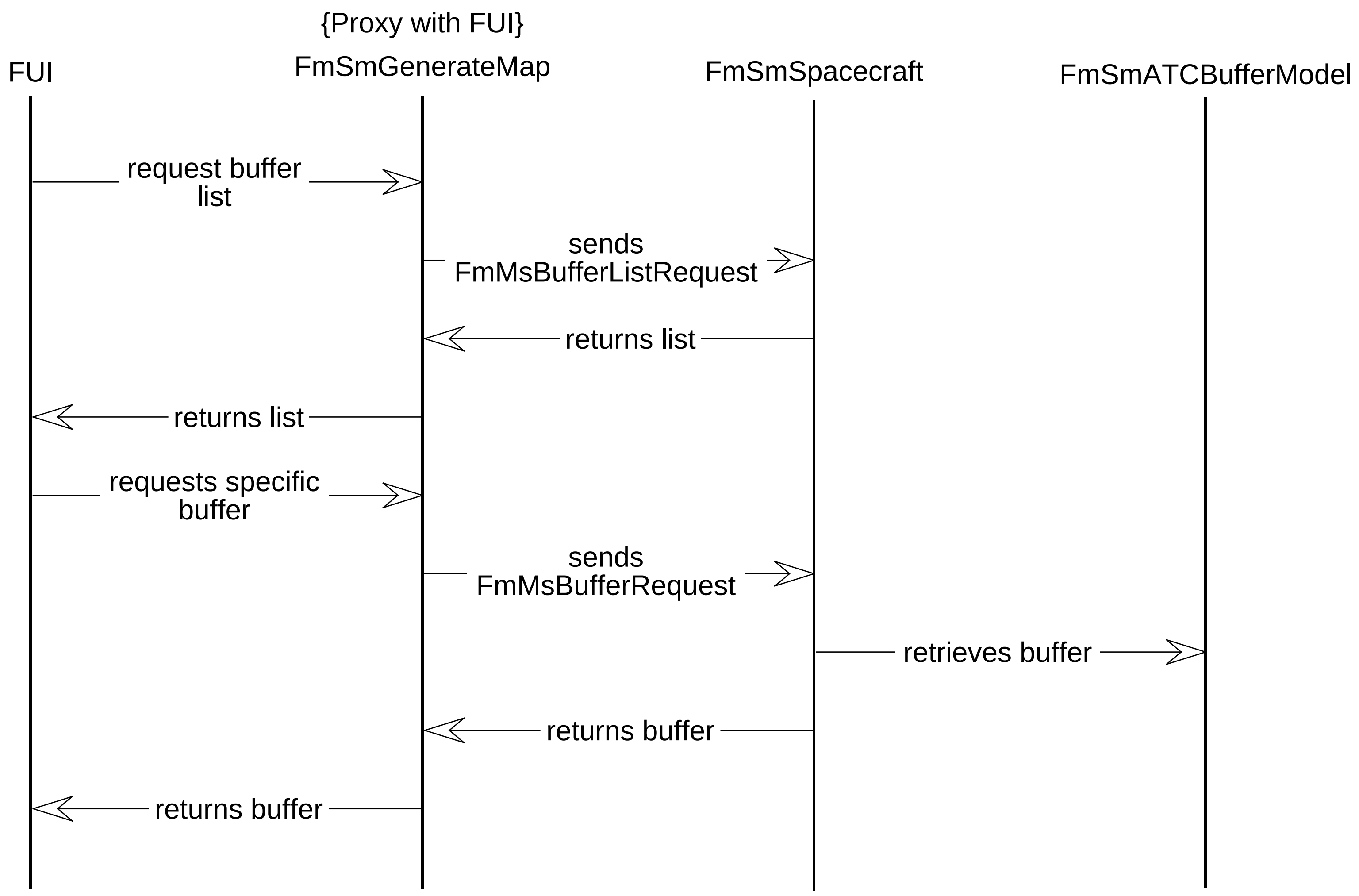


Figure 3.5-12. Spacecraft Model ATC Buffer Display Event Trace

3.5.4.5.2 ATC Buffer Display Summary Information

Interfaces:

- FUI

Stimulus:

- Receipt of Buffer List Request
- Receipt of Buffer Request

Desired Response:

- Buffer list is returned for display
- Buffer is returned for display

Pre-Conditions:

- Spacecraft Model software has been initiated

Post-Conditions:

- None

3.5.4.5.3 ATC Buffer Display Scenario Description

FUI sends requests via the FmMsGenerateMap proxy. The ATC buffer display will first request a list of available ATC buffers, then it will request a specific buffer for display. The ATC buffer list is the list of predicted and actual status ATC buffers.

3.5.4.6   RTS Buffer Display Scenario

3.5.4.6.1 RTS Buffer Display Scenario Abstract

The Spacecraft Model receives the requests to provide a buffer list and to provide a specified buffer based on a selection from that list (see Figure 3.5-13). This information is used by FUI to provide the RTS buffer display for the user.

3.5.4.6.2 RTS Buffer Display Summary Information

Interfaces:

- FUI

Stimulus:

- Receipt of Buffer List Request

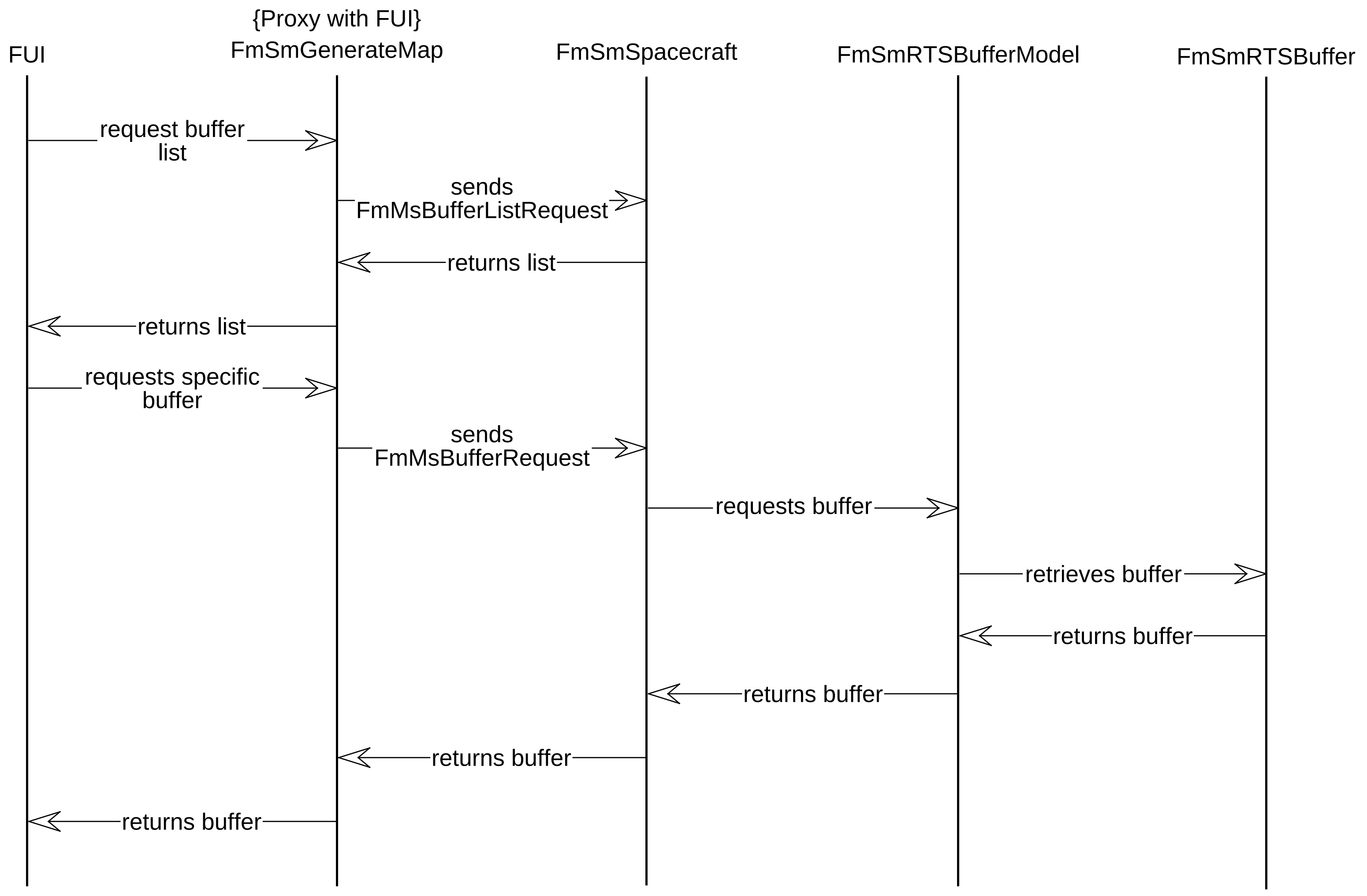


Figure 3.5-13. Spacecraft Model RTS Buffer Display Event Trace

- Receipt of Buffer Request

Desired Response:

- Buffer list is returned for display
- Buffer is returned for display

Pre-Conditions:

- Spacecraft Model software has been initiated

Post-Conditions:

- None

3.5.4.6.3 RTS Buffer Display Scenario Description

FUI sends requests via the FmMsGenerateMap proxy. The RTS buffer display will first request a list of available RTS buffers, then it will request a specific buffer for display. This list of available RTS buffers is all of the 128 AM-1 RTS buffers.

3.5.4.7 Map Report Generation Scenario

3.5.4.7.1 Map Report Generation Scenario Abstract

The Spacecraft Model receives the request to generate a map report (see Figure 3.5-14).

3.5.4.7.2 Map Report Generation Summary Information

Interfaces:

- FUI
- DMS

Stimulus:

- Receipt of Map Report Generation Request

Desired Response:

- Map Report is generated for specified Model (ATC, RTS or Table)

Pre-Conditions:

- Spacecraft model software has been initiated

Post-Conditions:

- Report is generated and stored in requested location.

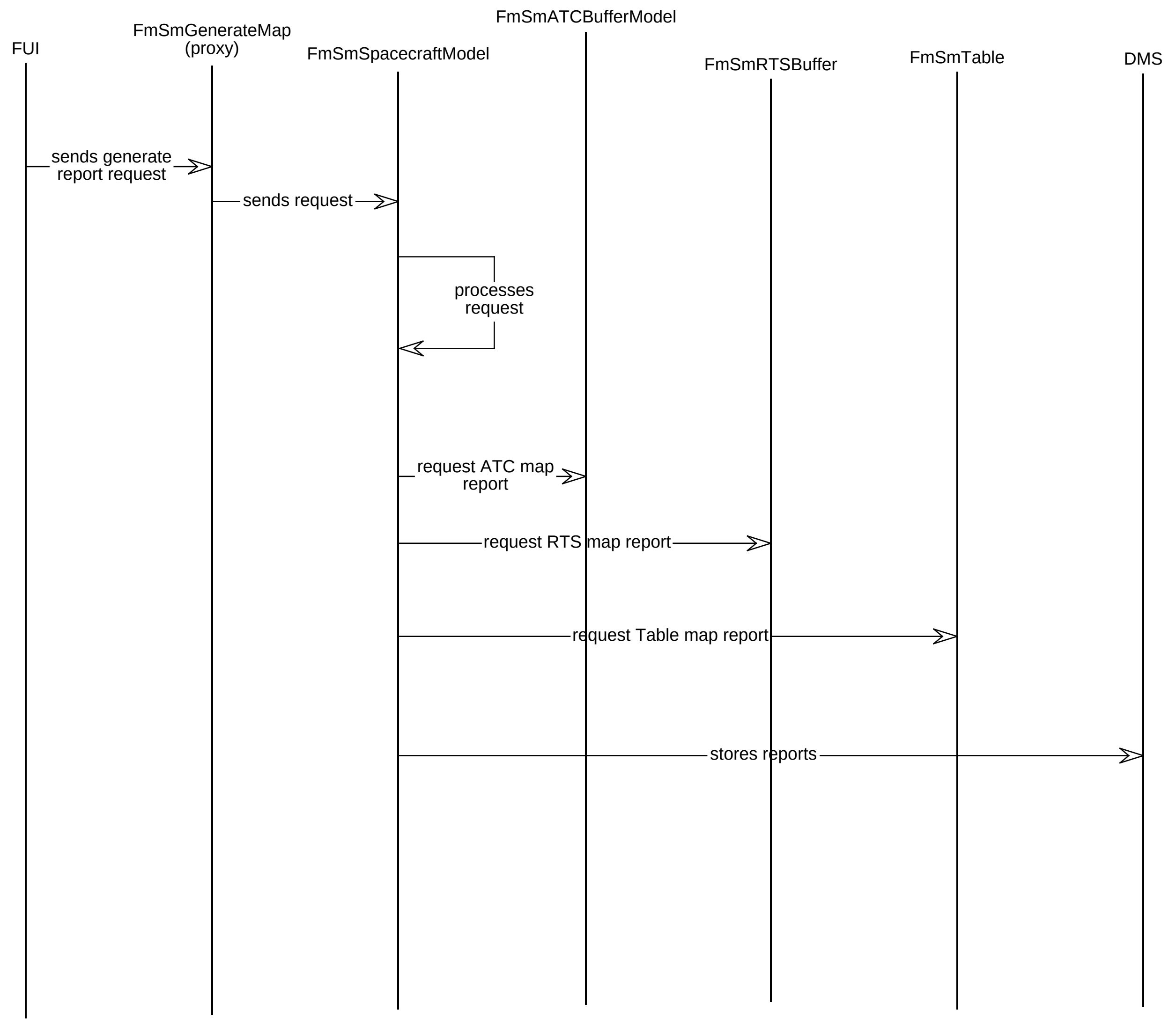


Figure 3.5-14. Spacecraft Model Map Report Event Trace

3.5.4.7.3 Map Report Generation Scenario Description

FUI send a request to generate a Map report via the FmMsGenerateMap proxy. The FmSmSpacecraftModel model requests the ATC buffer model, RTS buffer model or Table model to produce the report.

3.5.4.8 Image Report Generation Scenario

3.5.4.8.1 Image Report Generation Scenario Abstract

The Spacecraft Model receives the request generate an image report (see Figure 3.5-15).

3.5.4.8.2 Image Report Generation Summary Information

Interfaces:

- FUI

Stimulus:

- Receipt of Image report request

Desired Response:

- Image report is generated for specified ground image (ATC, RTS, or Table)

Pre-Conditions:

- Spacecraft model software has been initiated

Post-Conditions:

- None

3.5.4.8.3 Image Report Generation Scenario Description

FUI sends an Image report request via the FmMsGenerateMap proxy. Spacecraft model retrieves the proper image and produces the report.

FUI sends an Image overwrite request to the Spacecraft model via the FmMsGenerateMap proxy. The appropriate ground image is overwritten with the specified dump image.

3.5.4.9 Compare Report Generation Scenario

3.5.4.9.1 Compare Report Generation Scenario Abstract

The spacecraft model receives the request generate a comparison report (see Figure 3.5-16). The compare can be performed on any combination of images: load and ground.

3.5.4.9.2 Compare Report Generation Summary Information

Interfaces:

- FUI

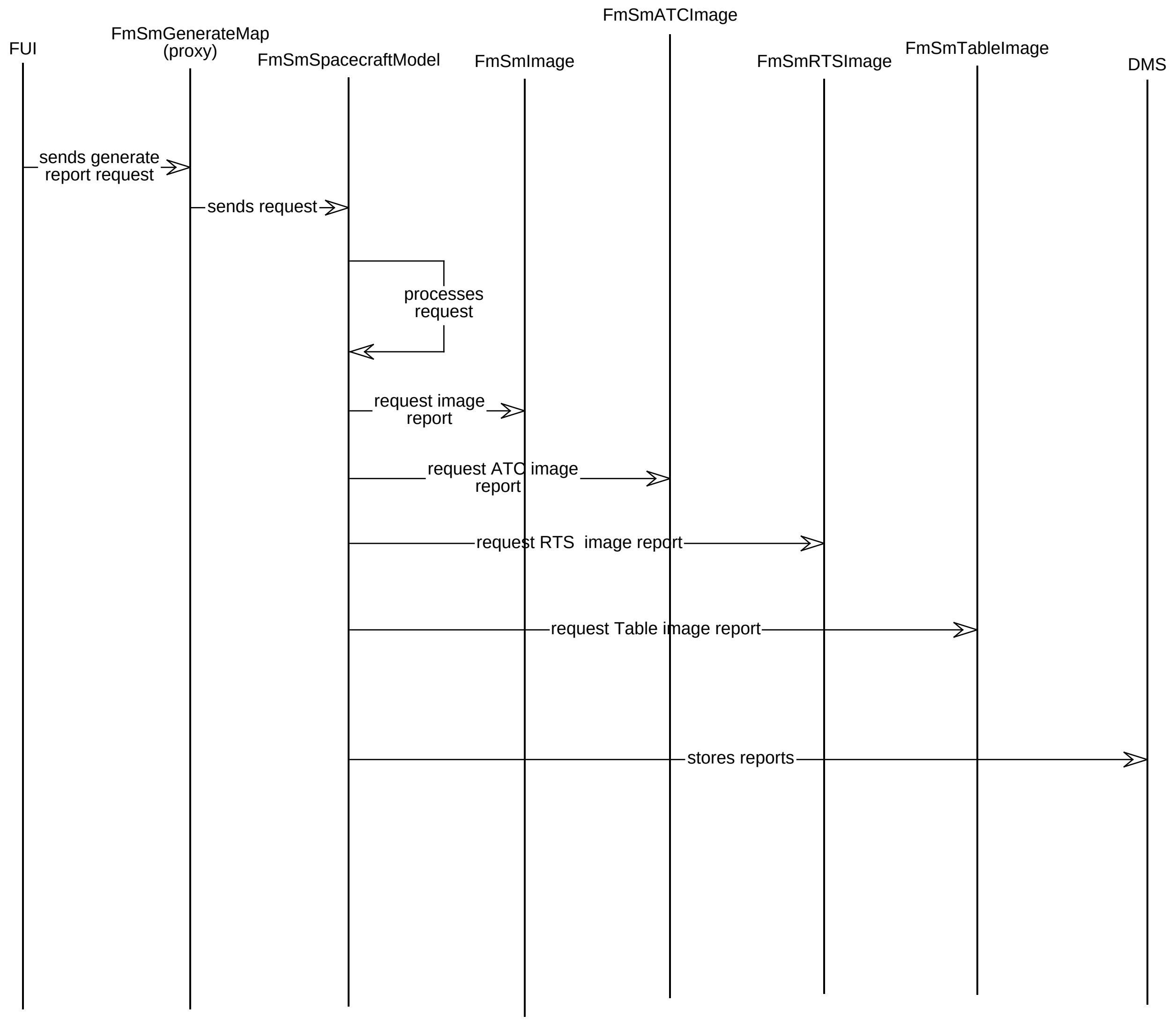


Figure 3.5-15. Spacecraft Model Image Report Event Trace

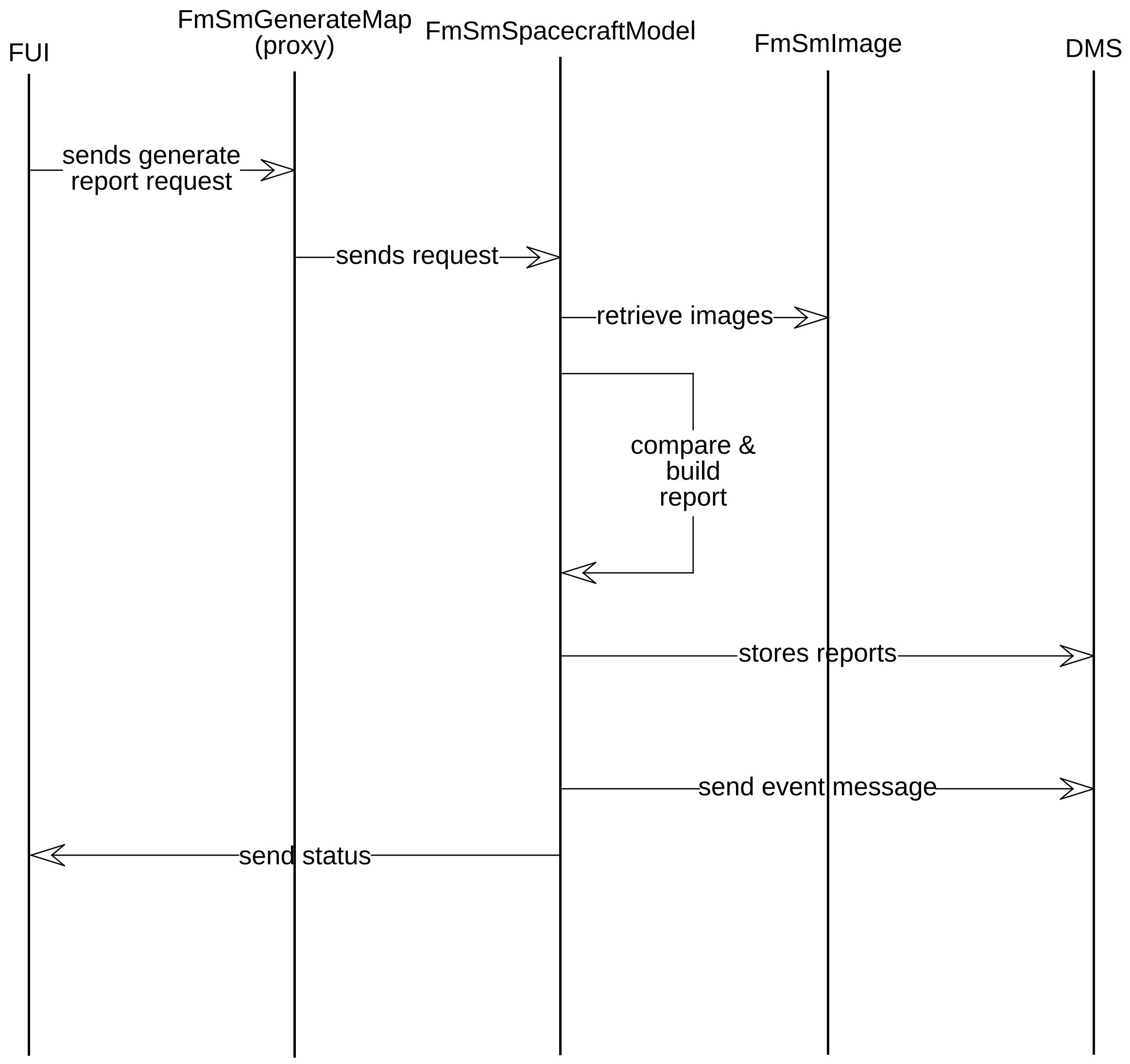


Figure 3.5-16. Spacecraft Model Compare Report Event Trace

Stimulus:

- Receipt of Compare request

Desired Response:

- Compare report is generated

Pre-Conditions:

- Spacecraft model software has been initiated

Post-Conditions:

- Compare report is generated and stored in requested location.

3.5.4.9.3 Compare Report Generation Scenario Description

FUI sends a Compare request to the Spacecraft model via the FmMsGenerateMap proxy. FmSmSpacecraftModel compares two image files and produces a report.

3.5.4.10 Table Model & Image Update Scenario

3.5.4.10.1 Table Model & Image Update Scenario Abstract

The spacecraft model receives the update buffer message from the CMS Load Catalog, queries the Table model to retrieve the correct table and updates the table model with the load contents (see Figure 3.5.17). Once the table model is updated the ground image is updated to reflect the new data. The Ground Image retrieves the correct table image and updates it with the specified table load image file.

3.5.4.10.2 Table Model & Image Update Summary Information

Interfaces:

- CMS Load Catalog

Stimulus:

- Receipt of Update Buffer Request

Desired Response:

- Table model is updated to reflect load content data field values
- Table ground image is updated

Pre-Conditions:

- Load Catalog software has been initiated
- Spacecraft model software has been initiated

Post-Conditions:

- None

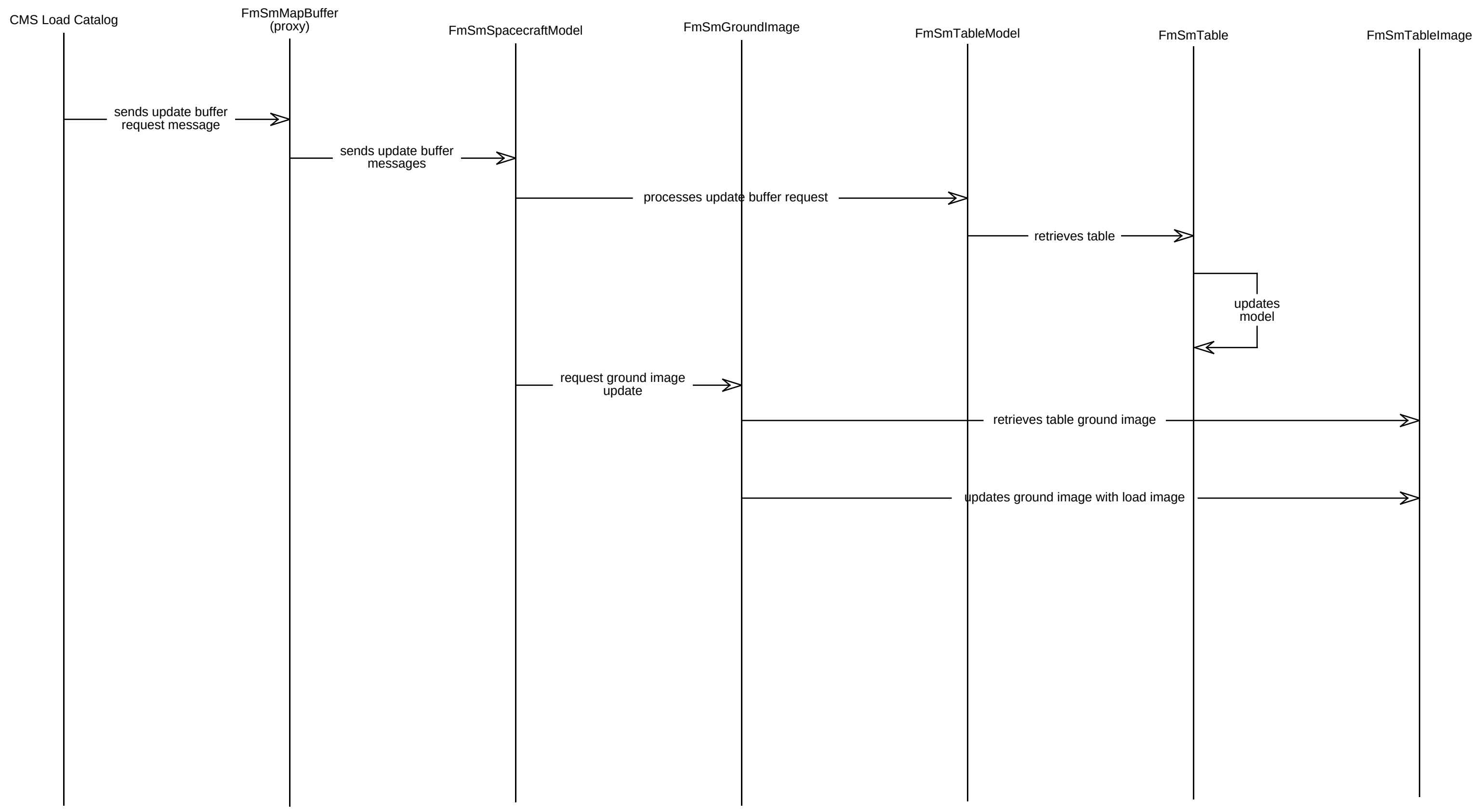

CMS Load Catalog
FmSmMapBuffer (proxy)
FmSmSpacecraftModel
FmSmGroundImage
FmSmTableModel
FmSmTable
FmSmTableImage
sends update buffer request message
sends update buffer messages
processes update buffer request
retrieves table
updates model
request ground image update
retrieves table ground image
updates ground image with load image

Figure 3.5-17. Spacecraft Model Table Model & Image Update Event Trace

3.5.4.10.3 Table Model & Image Update Scenario Description

The table model & image update scenario describes the receipt and processing of an update buffer request from CMS Load Catalog via FmSmMapBuffer proxy. When the load is successfully uplinked the table model is updated. The Spacecraft Model receives a request via FmSmMapBuffer to determine the table that the uplink load affects. Once the table model is updated, the table ground image is updated. FmSmGroundImage retrieves the appropriate table and updates the image with the load image file produced by FoLiLoad.

3.5.4.11 RTS Buffer Model & Image Update Scenario

3.5.4.11.1 RTS Buffer Model & Image Update Scenario Abstract

The spacecraft model receives the update buffer message from the CMS Load Catalog, queries the RTS model to retrieve the correct RTS buffer and updates the RTS buffer with the load contents, which is the directive list (see Figure 3.5-18). Once the RTS buffer is updated the ground image is updated to reflect the new data. The Ground Image retrieves the correct RTS buffer image and updates it with the specified RTS load image file.

3.5.4.11.2 RTS Buffer Model & Image Update Summary Information

Interfaces:

- CMS Load Catalog

Stimulus:

- Receipt of Update Buffer Request

Desired Response:

- RTS Buffer model is updated to reflect load content
- RTS buffer ground image is updated

Pre-Conditions:

- Load Catalog software has been initiated
- Spacecraft model software has been initiated

Post-Conditions:

- None

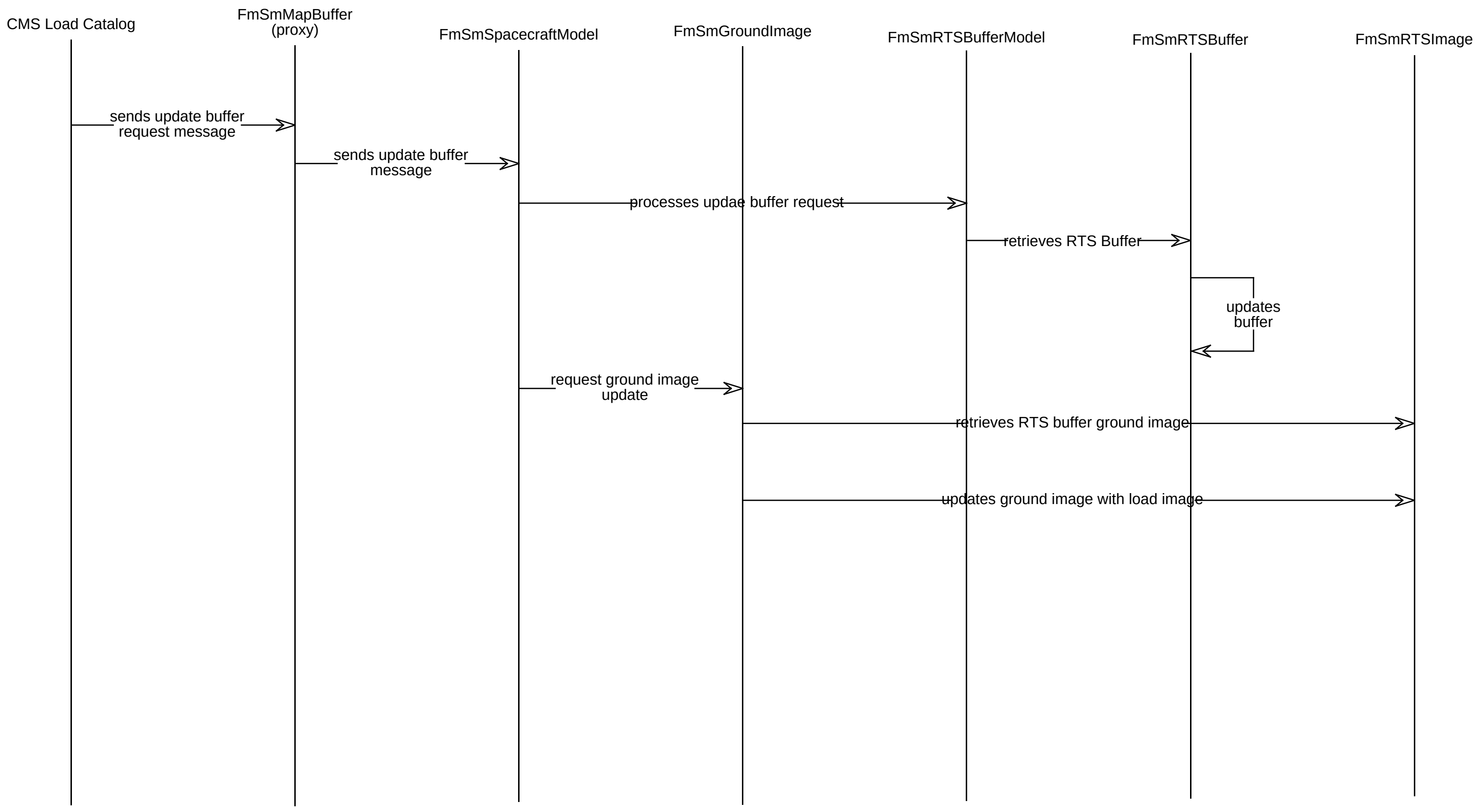

CMS Load Catalog
FmSmMapBuffer (proxy)
FmSmSpacecraftModel
FmSmGroundImage
FmSmRTSBufferModel
FmSmRTSBuffer
FmSmRTSImage
sends update buffer request message
sends update buffer message
processes updae buffer request
retrieves RTS Buffer
updates buffer
request ground image update
retrieves RTS buffer ground image
updates ground image with load image

Figure 3.5-18. Spacecraft Model RTS Model & Image Update Event Trace

3.5.4.11.3 RTS Buffer Model & Image Update Scenario Description

The RTS buffer model & image update scenario describes the receipt and processing of an update buffer request from CMS Load Catalog via FmSmMapBuffer proxy. When the load is successfully uplinked the RTS buffer model is updated. The Spacecraft Model receives a request via FmSmMapBuffer to determine the RTS buffer that the uplink load affects. Once the RTS buffer is updated, the RTS buffer ground image is updated. FmSmGroundImage retrieves the appropriate RTS buffer and updates the image with the load image file produced by FoLiLoad.

3.5.12.1 Table Data Request Scenario

3.5.12.1.1 Table Data Request Scenario Abstract

The spacecraft model receives the Table Data Request from the FUI, retrieves the table dump and table format from DMS, reverse-engineers the dump data into load contents and returns the load contents to FUI (see Figure 3.5-19).

3.5.12.1.2 Table Data Request Summary Information

Interfaces:

- FUI
- DMS

Stimulus:

- Receipt Table Data Request

Desired Response:

- Table dump is reverse-engineered to table load contents

Pre-Conditions:

- Schedule controller software has been initiated

Post-Conditions:

- Table load contents are created for use in FUI's table builder

3.5.4.12.3 Table Data Request Scenario Description

The table data request receipt scenario describes the receipt and processing of a FoMsTableDataReq from FUI via FmSmGenerateMap proxy (see Figure 3.5-20). The Spacecraft Model receives the request via FmSmGenerateMap to determine the table dump that is used to create the load contents. The EDU header information is stripped from the Dump data, then the binary dump data is extracted according to the FoFmTableFormat retrieved from DMS. The load contents are created and returned to FUI.

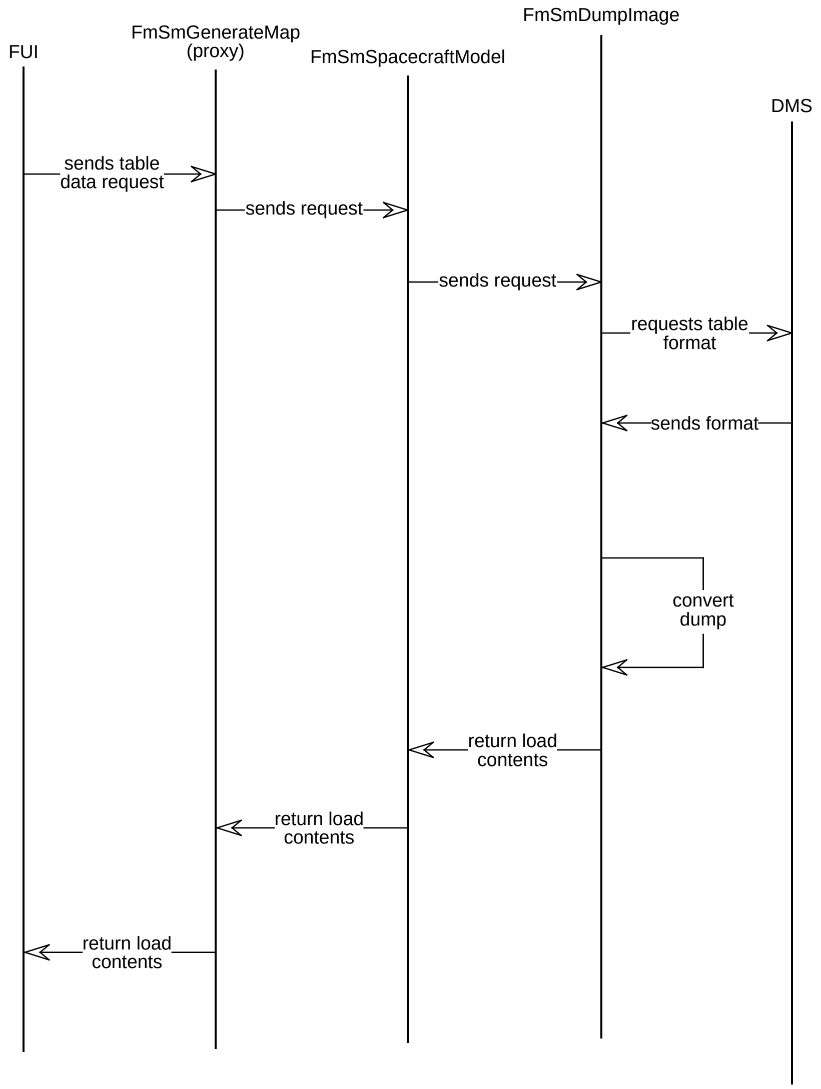
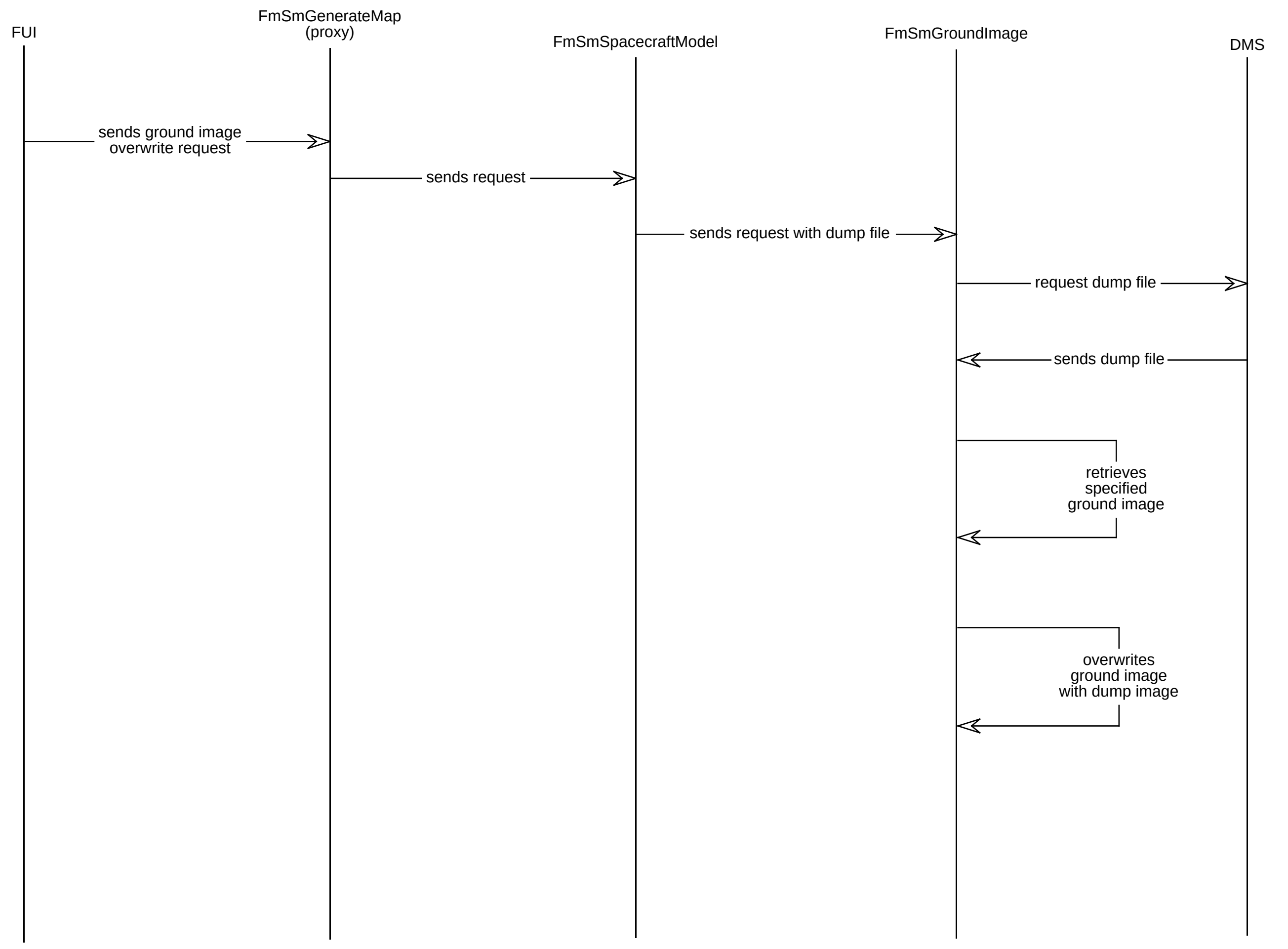


Figure 3.5-19. Spacecraft Model Table Data Request Event Trace


FUI
FmSmGenerateMap (proxy)
FmSmSpacecraftModel
FmSmGroundImage
DMS
sends ground image overwrite request
sends request
sends request with dump file
request dump file
sends dump file
retrieves specified ground image
overwrites ground image with dump image

GROUND IMAGE OVERWRITE

Figure 3.5-20. Spacecraft Model Ground Image Overwrite Event Trace

3.5.4.13 Ground Image Overwrite Receipt Scenario

3.5.4.13.1 Ground Image Overwrite Receipt Scenario Abstract

The spacecraft model receives the ground image overwrite request from the FUI, overwrites the specified ground image with the load image.

3.5.4.13.2 Ground Image Overwrite Summary Information

Interfaces:

- FUI

Stimulus:

- Receipt of Ground Image Overwrite request

Desired Response:

- Change ground image to reflect request

Pre-Conditions:

- Schedule controller software has been initiated

Post-Conditions:

- Ground Image is overwritten

3.5.4.13.3 Ground Image Overwrite Scenario Description

The spacecraft model receives the ground image overwrite request, FoMsImageOverWrite from the FUI via FmMsGenerateMap proxy. FmSmGroundImage retrieves the specified ground image: either the ATC image, one of the RTS images or one of the Table images. FmSmGroundImage then overwrites the specified ground image with the either the specified load image or the specified dump image.

3.5.5 Spacecraft Model Data Dictionary

Preprocessor Macros

`_FmMsATCBufferInfo_h_`

Types

`class` **`FmMsATCBufferInfo`**

This class represent the information returned to the proxy upon the complete processing of a GetATCBufStartTime. The time and the DAS id list are used to determine the commands necessary for constraint checking

Public Functions

`RWSlistCollectables&` **`GetDASList`**`(EcTVoid)`

Gets myDASIdList

`FoEcTime&` **`GetTime`**`(EcTVoid)`

Gets myTime

`EcTVoid` **`SetDASList`**`(const RWSlistCollectables&)`

Sets myDASList

`EcTvoid` **`SetTime`**`(const FoEcTime&)`

Sets myTime

Private Data

`RWSlistCollectable` **`myDASIdList`**

The list of DAS id's associated with the buffer being used to construct the new buffer for the new DAS processing request

`FoEcTime` **`myTime`**

The time of the 1st command in the buffer used to start the list for constraint checking.

Preprocessor Macros

`_FmMsBufferListRequest_h_`

Types

`class` **`FmMsBufferListRequest`**

Public Functions

`EcTInt` **`GetType`**`(EcTVoid)`

Gets myLoadType

`EcTVoid` **`SetType`**`(const EcTInt)`

Sets myLoadType

Private Data

`LoadType` **`myLoadType`**

Type of buffer list to be returned

`enum` **`LoadType`**

Enumeration of load types

Enumerators

ATC
RTS

Preprocessor Macros

_FmMsBufferRequest_h_

Types

class **FmMsBufferRequest**

This is a request for a specific buffer.

Protected Functions

EcTInt **GetBufferId**(EcTVoid)

GetsmyRTSBufferID

EcTInt **GetBufferType**(EcTVoid)

Gets myBufferType

RWCString& **GetName**(EcTVoid)

Gest myLoadName

EcTVoid **SetBufferId**(const EcTInt)

Sets myRTSBufferId

EcTVoid **SetBufferType**(const EcTInt)

Sets myBufferType

EcTVoid **SetName**(const RWCString&)

Sets myLoadName

Private Data

enum **myBufferType**

RWCString& **myLoadName**

If the buffer type is ATC the load name is need to retrieve the buffer

EcTInt **myRTSBufferId**

If the buffer type is RTS the RTS buffer id is need to retrieve the buffer

Private Types

enum

The buffer type being requested

Enumerators

ATC
RTS

Preprocessor Macros

_FmMsDeleteATCBuffers_h_

Types

class **FmMsDeleteATCBuffers**

This class represents a request to delete buffers It is only used when a late change is processed.

Protected Functions

`RWSlistCollectable& **GetList**(EcTVoid)`

Gets myLoadNames

`EcTVoid **SetList**(const RWSlistCollectables&)`

Sets myLoadNames

Private Data

`RWSlistCollectable **myLoadNames**`

The load names that are associated with the buffers, its is uses as a buffer id

Preprocessor Macros

_FmMsGenerateMap_h_

Types

class **FmMsGenerateMap**

This class represents the interface proxy class between FUI and the CMS FmSmSpacecraft class. FmSmSpacecraft manages the modelling for the ATC buffer, the RTS buffer, the table buffers and the ground imaging.

Public Functions

`EcTInt **CreateConnection**(void)`

Establishes a connection with FmSmSpacecraft to receive the Generate map report request.

`EcTVoid **DestroyConnection**(void)`

Destroyes the connections with FmSmSpacecraft

`EcTVoid **GenerateCompareReport**(EcTVoid)`

FUI invokes this function to generate a Compare report

`FoMsCMSStatus& **GenerateImageReport**(const FoMsImageRptReq&)`

FUI invokes this function to generate an Image report

`FoMsCMSStatus& **GenerateMapReport**(const FoMsGenMapRequest&)`

FUI invokes this function to generate an ATC or RTS map report

`RWSlistCollectables& **GetATCBuffers**(EcTVoid)`

FUI invokes this function to retrieve all the ATC buffers

`RWSlistCollectables& **GetRTSBuffers**(EcTVoid)`

FUI invokes this function to retrieve all the RTS buffers

`FoLiLoadContents&` **`ImportTableDumpt`**`(const FoMsTableDataReq&)`

ImportTableDump is called to retrieve the table dump and convert it into a table load contents file for editing

`foMsCMSStatus&` **`OverwriteGroundImage`**`(const FoMsImageOverWrite&)`

ground image be overwritten with the specified portion of the input image

`FoMsCMSStatus&` **`Receive`**`(void)`

Receives the results of the report generation

`RWCollectable&` **`RequestBuffer`**`(const FmMsBufferRequest&)`

RetrieveATCBuffer and RetrieveRTSBuffer call this function to request the FmSmSpacecraft model to return the appropriate buffer

`RWSlistCollectables&` **`RequestBufferList`**`(const FmMsBufferListRequest&)`

GetATCBuffers and GetRTSBuffers call this function to request the appropriate list of buffers from FmSmSpacecraft model

`FmSmATCBufferModel&` **`RetrieveATCBuffer`**`(const RWCString&)`

FUI invokes this function to request a specific ATC buffer

`FmSmRTSBuffer&` **`RetrieveRTSBuffer`**`(const EcTInt&)`

FUI invokes this function to request specific RTS buffer

`FoMsCMSStatus&` **`Send`**`(const RWCollectable&)`

Sends the report request to FmSmSpacecraft

Preprocessor Macros

_FmMsLoadData_h_

Types

class **FmMsLoadData**

This class is sent to CMS Schedule controller. From this class the ATC load directives are used to create the ATC binary uplink load. If the DAS needs to be partitioned multiple FmMsLoadData objects are returned to CMS Schedule Controller.

Protected Functions

`EcTInt` **`GetAddr`**`(EcTVoid)`

Returns the address of the next command in the directive list that is being processed

`FmMnDirectiveList&` **`GetDirectiveList`**`(EcTVoid)`

Returns the directive to be used to create the ATC load

`RWCString&` **`GetLoadName`**`(EcTVoid)`

Returns the Load name for the ATC load

`FmMnDirectiveList&` **`GetUplinkWindow`**`(EcTVoid)`

Returns the uplink window for the ATC load directive list

`EcTVoid` **`SetAddr`**`(const EcTInt)`

Sets the address to the next directive in the directive list that is being processed

`EcTVoid` **`SetDirectiveList`**`(const FmMnDirectiveList&)`

Sets myDirectiveList to the list used for creating the ATC load

`EcTVoid` **`SetLoadName`**`(const RWCString&)`

Sets the load name

`EcTVoid` **`SetUplinkWindow`**`(const FOSTimeInterval&)`

Sets the uplink window

Private Data

`EcTInt` **`myDirListAddr`**

This is the next directive in the processing list. If the list is completely processed the is set to NULL. If the list requires further processing, that is the DAS/ ATC load need to be partitioned, it is set to the next directive in the list. This is where the partitioned load needs to begin.

`FmMnDirectiveList` **`myDirectiveList`**

This is the portion of the DAS/ATC directive list being currently processed that will be used to create the ATC binary uplink load It may be all of the DAS or part of the DAS if the ATC buffer cannot hold all of the commands - that is the DAS is being partitioned

`RWCString` **`myLoadName`**

This is the load name create by ATC buffer model

`FOSTimeInterval` **`myUplinkWindow`**

This is the uplink window for the ATC uplink directive list

Preprocessor Macros

`_FmMsUpdateBuffer_h_`

Types

`class` **`FmMsUpdateBuffer`**

This class represents a message from the CMS load catalog to update the ATC/RTS buffers. It is sent from the load catalog when an uplink verification is received from the Command Subsystem It is also used to update the ground image.

Public Functions

`RWCString&` **`GetName`**`(EcTVoid)`

Returns the load name that is associated with the ATC load buffer

`EcTInt` **`GetType`**`(EcTVoid)`

Returns the type of buffer to be updated, ATC or RTS

`EcTVoid` **`SetName`**`(const RWCString&)`

Sets myLoadName

`EcTVoid` **`SetType`**`(const EcTInt)`

Sets the type of buffer to be updated, myType

Private Data

`EcTInt` **`myBufferID`**

Represent the buffer number that needs to be updated, pertains to RTS buffer number

`EcTInt` **`myEndLocation`**

Represent the End location in the buffer

`RWCString` **`myLoadName`**

Represents the load that was uplinked

`EcTInt` **`myStartLocation`**

Represents the start location in the buffer

RWCString **myTableName**

Represents the the table name for the buffer update

EcTInt **myType**

represents the type of buffer affected by the uplink

Preprocessor Macros

_FmSmATCBufferModel_h_

Types

class **FmSmATCBufferModel**

Represent a buffer model of the ATC buffer. There are multiple buffer models. The buffer models are used to determine the starting directive to be used for constraint checking. For this purpose a "previous" buffer models are kept. This previous buffer model will be used to constraint check late changes. Once all the commands in the buffer model are executed, the buffer is archived and removed from the active list of ATC buffer models.

There is one "actual" buffer model. This represents what is currently loaded to the spacecraft. When an FmMsUpdateBuffer is received by FmSmSpacecraft the "actual" buffer model is moved to the "previous" buffer model.

Finally, there are multiple "predicted" buffer models. These are used for constraint checking also. Because CMS performs most of its functionality in advance we are predicting what we expect the actual buffer model to be and using the "most recent predicted buffer model" for constraint checking and determining what the new buffer model will be. When an FmMsUpdate-Buffer is received by FmSmSpacecraft the "predicted" buffer model with the specified load name is moved to the "actual" buff-er model.

Public Functions

EcTVoid **AddSafeCommands**(const FmMnDirectiveList&)

Adds the safe commands to the end of the buffer and load directive list

FmMsLoadData& **AssignCommandLocations**(const FmMnDirectiveList&)

Assigns the directives to a buffer location

EcTInt **BuildBuffer**(const FOSTimeInterval&, FmMnDirectiveList&, EcTInt&, FmMsLoadData&)

Builds the buffer this is the controlling function used to determine the valid uplink window, available locations in the buff-er, keeps activities together and determines the commands for the load

EcTInt& **DetermineActForBuffer**(const EcTInt, const FmMnDirectiveList&)

This routine ensures that activities are not split for the load that is being built. When it is determined that no more "full" activities will fit in the buffer the load is marked for partitioning

EcTVoid **DetermineLoad**(FmMsLoadData&)

Determines all of the commands for the ATC load

EcTVoid **DeterminePartitionUplinkWindow**(const FoEcSpaceDirective&, FmMs-LoadData&)

determines uplink window for partition

EcTVoid **DetermineUplinkWindow**(const FOSTimeInterval&, const FoEcSpaceDi-rective&, FmMsLoadData&)

validates the requested uplink window

EcTVoid **GenerateMapReport**(EcTVoid)

Generate a map report for ATC buffer

EcTInt **LocationsAvailable**(const FoEcSpaceDirective&)

Determines the number of available buffer locations in the buffer

EcTInt **Partition**(FmMnDirectiveList&, EcTInt&, FmMsLoadData&)

Builds the buffer for a partition, this is the controlling function used to determine the valid uplink window for the partition, available locations in the buffer, keeps activities together and determines the commands for the load

EcTVoid **UpdateModel**()

Updates model from working to predicted or predicted to actual or from actual to previous

Private Data

RWSlistCollectable **myDASIdList**

List of DAS Id associated with the buffer

EcTInt **myEndLoc**

The location of the last executable command in the buffer

RWCString **myLoadName**

The Load name that identifies the buffer

EcTInt **myNumberofSafeCmds**

the number of safecommands for this buffer

RWSlistCollectable **mySafeCommands**

The list of safe commands that will be added to the end of the buffer

EcTInt **myStartLoc**

The location of the first executable command for the load/buffer

RWTime **myTime**

The time for determining the most recent buffer, it is set to the time of the DAS the buffer is being created for;

EcTInt **myType**

Indicates if the buffer is the actual buffer, a predicted buffer, a previous buffer, or a working buffer

FOSTimeInterval **myUplinkWindow**

the uplink window for the load associated with this buffer

Preprocessor Macros

_FmSmATCImage_h_

Types

class **FmSmATCImage**

ATC Dump Image

Preprocessor Macros

_FmSmCompareReport_h_

Types

class **FmSmCompareReport**

Generates a report on the compare of two dump files

Private Data

RWString **myReportName**

Name of the report to be generated

Include Files

`FmSmGroundImage.h`

Preprocessor Macros

`_FmSmDumpImage_h_`

Types

`class` **`FmSmDumpImage`**

Dump Image Class

Base Classes

`public` **`FmSmGroundImage`**

Public Functions

`EcTVoid` **`CompareDumpWithDefaults`**`(RWString)`

Compares the value of a table dump with the default values stored in DMS

`FoLiLoadContents` **`ConvertDumptoContents`**`(FoMsTableDataReq)`

Converts a dump file to an ASCII load contents file

`EcTVoid` **`GenerateReport`**`(FoMsImageRptReq)`

Generates a report on a given dump file

Preprocessor Macros

`_FmSmDumpReport_h_`

Types

`class` **`FmSmDumpReport`**

Public Functions

`RWString` **`GetReportName`**`(void)`
`EcTVoid` **`SetReportName`**`(RWString)`

Private Data

`RWString` **`myReportName`**

Preprocessor Macros

`_FmSmFSWImage_h_`

Types

`class` **`FmSmFSWImage`**

Flight Software image class

Preprocessor Macros

`_FmSmGroundImage_h_`

Types

`class` **`FmSmGroundImage`**

Public Functions

`EcTVoid` **`GenerateImageReport`**`(RWString, EcTInt)`

Generates a report from an dump image

Include Files

`FmSmGroundImage.h`

Preprocessor Macros

`_FmSmImage_h_`

Types

`class` **`FmSmImage`**

Base Classes

`public` **`FmSmGroundImage`**

Public Functions

`EcTVoid` **`GenerateReport`**`(FmMsImageRptReq)`

Generates a image report based on an input request

Preprocessor Macros

`_FmSmMapBuffer_h_`

Types

`class` **`FmSmMapBuffer`**

This class represents the interface proxy class between CMS internal subsystems and the FmSmSpacecraftModel class. FmSm-SpacecraftModel manages the buffer modelling for ATC, RTS and table buffers and the ground imaging.

Public Functions

`EcTInt` **`CreateConnection`**`(EcTVoid)`

Establishes a connection with FmSmSpacecraft to receive requests from the schedule controller and the load catalog

`EcTVoid` **`DeleteBuffers`**`(const RWSlistCollectables&)`

Request received from load catalog when a late change as been successfully processed. The predicted buffer models associated with all of the generated loads are deleted. Instantiates an FmMsDeleteATCBuffers object.

`EcTVoid` **`Destroy`**`(EcTVoid)`

Destroys the connection with FmCcCommandModel

`FmMsATCBufferInfo` **`GetATCBufStartTime`**`(const FoEcTime&)`

Requests the start time of the 1st command in the buffer that will be used to model the newly recieved DAS or late change request

RWSlistCollectables& **MapATC**(const FmMnDirectiveList&, const FOSTimeInterval&, const FoEcTime&, const EcTInt&)

Request FmSmSpacecraft to map the command list into an ATC buffer model. Instantiates an FmMsATCMapRequest object to be sent to FmSmSpacecraft.

RWSlistCollectables& **MapLateChange**(const FmMnDirectiveList&, const FOSTimeInterval&, const FoEcTime&, const EcTInt&)

Requests FmSmSpacecraft to map the late change command list into the correct buffer model. Instantiates an FmMsATCMapRequest object to be sent to FmSmSpacecraft.

RWSlistCollectables& **Receive**(EcTVoid)

Receives the response from FmSmSpacecraftModel It receives either A list of FmMsLoadData objects or a FmMsATCBufferInfo object

EcTVoid **Send**(const RWCollectable&)

Sends messages to FmSmSpacecraftModel. Sends FmMsATCMapRequest, FmMsDeleteATCBuffers, or FmMsUpdateBuffer.

EcTVoid **UpdateBuffer**(const FmMsUpdateBuffer&)

Request the buffer be updated to a new status

Preprocessor Macros

_FmSmMicroLoadImage_h_

Types

class **FmSmMicroLoadImage**

Microprocessor load image class

Preprocessor Macros

_FmSmRTSBuffer_h_

Types

class **FmSmRTSBuffer**

This class represents a single RTS buffer. It models the contents of the buffer: the contents of each location are maintained. A RTS map report may be generated from this class and authorization to access this class is verified using CSS software.

Public Functions

EcTVoid **GenerateMapReport**(EcTVoid)

Generate the RTS map report upon request from FUI

EcTVoid **UpdateBuffer**(const FmMsUpdateBuffer&)

Updates the RTS buffer when an FmMsUpdateBuffer message is received from CMS Load Catalog. The message is sent in response to the receipt of an uplink verification from R/T Command

EcTVoid **VerifyAuthorization**(EcTVoid)

Verifies authorization to the RTS buffer

Private Data

EcTInt **myBufferId**

Identifies the buffer being accessed

EcTInt **myCriticalFlag**

Idicates that the load contained critical commands, therefore making the buffer critical

`RWCString` **`myCurrentLoad`**

Indicates the load file name that was uplinked to this buffer

`EcTInt` **`myInhibitId`**

Indicates the id used to inhibit the commands in the buffer

Preprocessor Macros

`_FmSmRTSBufferModel_h_`

Types

`class` **`FmSmRTSBufferModel`**

This class represents the managing class of all the RTS buffers It will retrieve a requested buffer, update the specified buffer, and request the buffer to generate its map report.

Public Functions

`EcTVoid` **`GenerateMapReport`**`(const FoMsGenMapRequest&)`

Requests the RTS buffer specified in the request to generate its map report

`FmSmRTSBuffer&` **`RetrieveBuffer`**`(const EctInt)`

Retrieves the requested buffer number

`EcTVoid` **`UpdateModel`**`(const FmMsUpdateBuffer&)`

Updates the specified buffer model

Private Data

`EcTInt` **`myNumberOfBuffers`**

indicates the number of buffers being modelled

Preprocessor Macros

`_FmSmRTSImage_h_`

Types

`class` **`FmSmRTSImage`**

Preprocessor Macros

`_FmSmSpacecraftModel_h_`

Types

`class` **`FmSmSpacecraftModel`**

This is the controlling class for the spacecraft modelling. FmSmSpacecraftModel handles all of the interprocess communication between CMS Schedule controller, CMS Load Catalog, and FUI. It asks the models, ATC, RTS, and table to determine the mapping of the commands into the buffer locations.

The ATC buffer model determines the appropriate uplink window and if the load needs to be partitioned

Public Functions

`EcTVoid` **`ArchiveATCBuffer`**`(const RWCString, const EcTInt)`

buffer image once all the commands in the buffer have been executed it is stored with DMS, RTSs and Tables are archived when the actual buffer is being updated to a new buffer.

`FmSmImage` **`ConvertDumptoBinaryImage`**`(RWString)`

Converts a dump file into a binary image file by stripping out the EDU header and packet header information

`RWSlistCollectable&` **`CreateATCBuffer`**`(const FmMsATCMapRequest&)`

Creates a new ATC working buffer, the working buffer will be moved into the predicted buffer list after the ATC load is generated

`EcTVoid` **`DeleteATCBuffers`**`(const FmMsDeleteATCBuffers&)`
`FmSmATCBufferModel&` **`GetRecentBuffer`**`(const FoEcTime&)`

Deletes ATC buffers from the list

`EcTVoid` **`HandleMessage`**`(const RWCollectable&)`

Handles all IPC messaging

`EcTVoid` **`Initialize`**`(EcTVoid)`

Initializes the spacecraft modelling process

`FoMsCMSStatus&` **`ProcessCompareRequest`**`(const FmMsCompareRequest&)`

Processes a Compare request

`FoMsCMSStatus&` **`ProcessImageReport`**`(const FoMsImageRptReq&)`

Processes a request to produce an image report

`EcTVoid` **`ProcessMapReq`**`(EcTVoid)`

Prcesses a generate map request

`EcTVoid` **`ProcessMemoryDump`**`(EcTVoid)`

Processes a memory dump request

`FmSmATCBufferModel&` **`RetrieveATCBuffer`**`(const RWCSting&)`

Retrieves the requested ATC buffer

`RWCollectable&` **`RetrieveBuffer`**`(const FmMsBufferRequest&)`

Retrieves requested buffer - either ATC or RTS

`RWSlistCollectables&` **`RetrieveBufferList`**`(const FmMsBufferListRequest&)`

Retrieves a list of buffers - RTS or ATC

`EcTVoid` **`SendEventMessage`**`(const RWCString&)`

Sends an event message to DMS for broadcasting

`EcTVoid` **`UpdateATCModel`**`(const RWCString&)`

Updates the ATC model - moves the working ATC model into a predicted model or a predicted model to the actual

`EcTVoid` **`UpdateBuffer`**`(const FmMsUpdateBuffer&)`

Updates appropriate buffer model

`EcTVoid` **`UpdateRTSModel`**`(const FmMsUpdateBuffer&)`

Updates the appropriate RTS buffer

`EcTVoid` **`UpdateTableModel`**`(const FmMsUpdateBuffer&)`

Updates the Table model

Preprocessor Macros

`_FmSmTable_h_`

Types

class **FmSmTable**

Represent a single table on the spacecraft, it will update the ground representation of the spacecraft tables.

Public Functions

`EcTVoid` **`GenerateMapReport`**`(EcTVoid)`

Generates the table map report

`EcTVoid` **`UpdateTable`**`(const FmMsUpdateBuffer&)`

Updates the ground reference table when an FmMsUpdatebuffer message is received from CMS Load Catalog. The message is sent in response to the receipt of an uplink verification from R/T Command

Private Data

`RWCString` **`myCurrentLoad`**

indicates the load that was uplinked and used to update the buffer

`EcTInt` **`myEndingLocation`**

the Ending location of the buffer

`RWCString` **`myOwner`**

the owner of the table

`EcTInt` **`mySize`**

the size of the table

`EcTInt` **`myStartLocation`**

the starting location of the table

`FoLiTableFormat` **`myTableFormat`**

The table format for this table

Preprocessor Macros

`_FmSmTableImage_h_`

Types

class **FmSmTableImage**

Table Image Class

Private Data

`RWCString` **`myName`**

Name of this table image

Preprocessor Macros

`_FmSmTableModel_h_`

Types

class **FmSmTableModel**

This class represents themanaging class of all the table buffers It will retrieve a requested table, update the specified table with the uplink load, and request the table to generate its map report

Public Functions

`EcTVoid` **`GenerateMap`**`(const FoMsGenMapRequeset&)`

Requests the Table specified in the request to genereate ist map report

`FmSmTable&` **`RetrieveTable`**`(const RWCStrin&)`

Retrieves the requested table

`EcTVoid` **`UpdateModel`**`(const FmMsUpdateBuffer&)`

Updates the specified table model

System Include Files

`rw/collect.h`

Preprocessor Macros

`_FoFmDataField_h_`

Types

class **FoFmDataField**

This class represents the data field of a table

Base Classes

`public` **`RWCollectable`**

Public Functions

`RWBitVec` **`ProduceBinary`**`()`

Produces the binary form of this field.

Private Data

`RWCString` **`myDataUnits`**

The units of the field value.

`RWCString` **`myFieldDescriptor`**

Textual information describing the field and its value.

`EcTInt` **`myFieldNumber`**

A unique value which identifies the field within the table.

`EcTInt` **`myRangeCheckFlag`**

An indicator of whether range checking is to be performed.

`EcTInt` **`myScaleFactor`**

The scale factor to be applied to the word value within this field.

EcTInt **myTableNumber**

A unique value specifying a memory table.

EcTInt **myValueBitSize**

The size of the value in bits.

EcTInt **myValueOverrideFlag**

An indicator of whether the value may br overwritten with a new value during table generation.

RWCString **myValueType**

The data type of the value in this field.

Preprocessor Macros

_FoMsCMSStatus_h_

Types

class **FoMsCMSStatus**

This class is used to return processing status to CMS's external interfaces. It returns status for constraint checking and for load generation.

Private Data

EcTInt **myId**

or the Instruction request id that the status is in response to

RWCString **myStatus**

The status is either:

```
complete - everything processed without error
pending -  the constraint check was complete with
           soft constraints only
failed - constraint violations found were hard constraints
         load generation failed
```

Preprocessor Macros

_FoMsCompareMask_h_

Types

class **FoMsCompareMask**

This class represents a user specified mask for the compare report. The compare can be requested for a ground image and a dump image or a load image or any combination of image comparisons. The mask specifies a particular portion of the image not to compare.

Private Data

EcTInt **myEndAddress**

the ending address of the mask

EcTInt **myStartAddress**

the starting address of the mask

Preprocessor Macros

`_FoMsCompareReq_h_`

Types

`enum` **`FileType`**

Enumeration of file types to be compared.

Enumerators

`ATC`

Absolute Time Command Buffer

`FSW`

Flight Software Buffer

`MP`

MicroProcessor Buffer

`RTS`

Relative Time Command Buffer

`TAB`

Table Buffer

`class` **`FoMsCompareReq`**

Private Data

`EcTInt` **`myEndAddress`**

This is the ending address for the comparison

`RWCString` **`myImageFile1`**

this is the first image file for use in the comparison

`RWCString` **`myImageFile2`**

this is the second image file for use in the comparison

`EcTInt` **`myStartAddress`**

This is the starting address for the comparison

`FileType` **`myType`**

the types of files to be compared

Preprocessor Macros

`_FoMsConflictInfo_h_`

Types

`class` **`FoMsConflictInfo`**

This class gives the identifying information on constraint violations. It specifies the id, the command mnemonic, the conflicting command, the time the constraint violation occurred, whether the violation is hard or soft and a textual description of the violaion

Private Data

RWCString **myCmdMnemonic**

the directive command mnemonic being constraint checked

RWCString **myConflictingCmd**

the command that violates the constraint rule

RWTime **myConstraintTime**

the time of the constraint

EcTInt **myId**

the ID represent different things for different constraint checking requests:

```
    - the activity id of a command in a schedule
    - the line number of a command in a procedure
    - the buffer location of a command in an RTS load
      contents file
    - the PDB activity definition id
```

EcTInt **mySoftHardFlag**

Indicates if the violatin is hard or soft

RWCString **myViolationInfo**

Textual description of the violation for messaging

Include Files

FoUiInstruction.h

Preprocessor Macros

_FoMsGenMapRequest_h_

Types

class **FoMsGenMapRequest**

This class is a request for a buffer map report

Base Classes

public **FoUiInstruction**

Private Data

EcTInt **myBufferID**

The RTS buffer number used for the map report

EcTInt **myEndLocation**

the ending buffer locatin for the report

RWCString **myLoadName**

the load name used to identify the atc buffer model

enum **myMapType**

EcTInt **myStartLocation**

the starting location in the buffer for the report

Private Types

`enum`

The type of buffer for which to generate the map report

Enumerators

`ATC`
`RTS`

Preprocessor Macros

`_FoMsImageOverWrite_h_`

Types

`class` **`FoMsImageOverWrite`**

This class represents a request to overwrite a ground with another (dump) image or a portion of that image.

Private Data

`RWCString` **`myDumpName`**

this is the name of the dump image to be used to overwrite the ground image

`EcTInt` **`myStartAddres`**

This is the starting address for the overwrite

`EcTInt` **`myStopAddress`**

This is the stop address for the overwrite

Preprocessor Macros

`_FoMsImageRptReq_h_`

Types

`class` **`FoMsImageRptReq`**

This class represents a request to output an image report

Private Data

`RWCString` **`myDirectory`**

this is the directory where the report is output

`RWCString` **`myImageName`**

This is the image file to produce the report from

`RWCString` **`myReportName`**

This is the report file

Include Files

`FoMsCMSStatus.h`

Preprocessor Macros

`_FoMsStatusComplete_h_`

Types

`class` **`FoMsStatusComplete`**

Represents a good status from constraint checking or load generation

Base Classes

`public` **`FoMsCMSStatus`**

Include Files

`FoMsCMSStatus.h`

Preprocessor Macros

`_FoMsStatusFailed_h_`

Types

`class` **`FoMsStatusFailed`**

Represents a failed status from constraint checking of load generation

Base Classes

`public` **`FoMsCMSStatus`**

Include Files

`FoMsCMSStatus.h`

Preprocessor Macros

`_FoMsStatusPending_h_`

Types

`class` **`FoMsStatusPending`**

Represents a pending status from constraint checking This means that the constraint violations found are all soft constraints. If CMS is processing a DAS, we wait to continue processing of the load until a response is received from planning and scheduling. If CMS is processing an RTS load we wait for a response from FUI to continue processing the RTS load.

Base Classes

`public` **`FoMsCMSStatus`**

Include Files

`FoUiInstruction.h`

Preprocessor Macros

`_FoMsTableDataReq_h_`

Types

`class` **`FoMsTableDataReq`**

This class represents a request from the FUI's table load builder to import a table dump and convert it to a table load contents for editing & producing a table load

Base Classes

`public` **`FoUiInstruction`**

Private Data

`String` **`myDirectory`**

This is the directory for the output report file

`String` **`myDump`**

Represents the dump file to be used for the conversion to a table load contents file

Preprocessor Macros

`_FoRpMapReport_h_`

Types

`class` **`FoRpMapReport`**

A description of the class

Preprocessor Macros

`_FoSmBufferLocation_h_`

Types

`class` **`FoSmBufferLocation`**

This class represnt the buffer location of a command in either the RTS or ATC buffers

Protected Functions

`FoEcSpaceDirective&` **`GetDirective`**`(EcTVoid)`

Gets mySpaceDirective

`EcTInt` **`GetLocation`**`(EcTVoid)`

Gets myLocation

`EcTVoid` **`SetDirective`**`(const FoEcDirective&)`

sets mySpaceDirective

`EcTVoid` **`SetLocation`**`(EcTInt)`

Sets myLocation

Private Data

`EcTInt` **`myLocation`**

the buffer location (0-2999)

`FoEcSpaceDirective` **`mySpaceDirective`**

The directive in myLocation

3.6 Load Catalog

The Load Catalog is a persistent process that runs on the FOS Data Server. It is responsible for generating loads and for maintaining a catalog of all valid loads that are available for uplink by the FOS.

Load Catalog generates loads from load contents files. Depending on the load type, Load Catalog performs whatever conversion is needed to format the load as 1553B commands, computes the CRC for the load, and builds and appends the load initiate command. The 1553B commands are packetized according to CCSDS protocol.

Microprocessor and flight software load contents files are received by the FOS as binary files and the generation of these types of loads only requires the conversion of the binary data to 1553B commands.

RTS load contents files are passed to Load Catalog from FUI. Load Catalog is responsible for requesting a constraint check on the command sequence by Command Model and for converting each command and its time tag to binary format. The commands are then converted to 1553B commands.

Table load contents files are passed to Load Catalog from FUI for user-generated tables, from DMS for FDF tables, and from FAS for the spacecraft clock correlation table. Load Catalog converts each field of the table from ASCII to binary using the table definition in the FOS database. The table data is then converted to 1553B commands.

ATC loads are created by the CMS based on an activity list received from PAS. After the command list generated from the activity list has been constraint checked and partitioned, the command list for an ATC load is passed to Load Catalog by the Schedule Controller process. Load Catalog converts each command in the list to its binary format, and converts it to a 1553B command.

For each load available for uplink by the FOS, Load Catalog maintains four files: the load contents from which the load was generated, the load in uplink format, the load generation report, and the load image that may be compared against a memory dump image. Load Catalog also maintains identifying information about the load as catalog entries in the FOS database. Whenever a new load is generated, Load Catalog stores the four files for that load with DMS and requests that DMS create a new catalog entry for the load. When a load is deleted by user request, Load Catalog removes the four files for that load with DMS and requests that DMS remove the catalog entry for the load. Load Catalog is also responsible for updating fields in the catalog entry when a load is scheduled for uplink or uplinked successfully.

3.6.1 Load Catalog Context

Figure 3.6-1 shows the context diagram for Load Catalog. Load Catalog has seven interfaces.

FOS Analysis:

- Load Catalog receives a request to generate a Table Load from a load contents file.

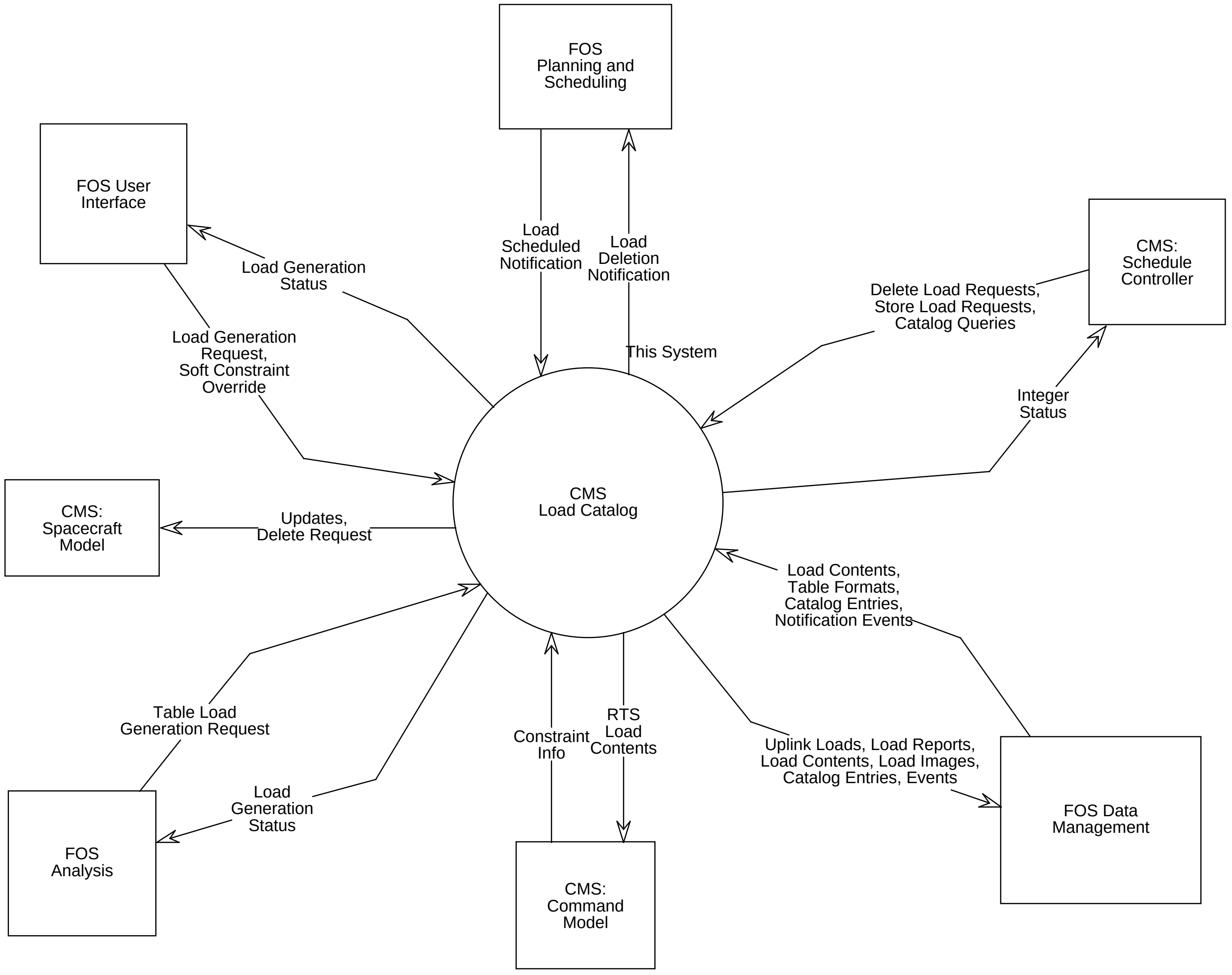

FOS Planning and Scheduling
FOS User Interface
CMS: Schedule Controller
Load Scheduled Notification
Load Deletion Notification
This System
Load Generation Status
Load Generation Request, Soft Constraint Override
Delete Load Requests, Store Load Requests, Catalog Queries
Integer Status
CMS Load Catalog
CMS: Spacecraft Model
Updates, Delete Request
Load Contents, Table Formats, Catalog Entries, Notification Events
Table Load Generation Request
Constraint Info
RTS Load Contents
Uplink Loads, Load Reports, Load Contents, Load Images, Catalog Entries, Events
Load Generation Status
FOS Analysis
CMS: Command Model
FOS Data Management

Figure 3.6-1. Load Catalog Context Diagram

- Load Catalog sends an integer status to indicate if the load was successfully generated or not.

CMS Spacecraft Model:

- Load Catalog sends requests to update buffer models whenever a load has been successfully uplinked.
- Load Catalog sends delete requests whenever loads are deleted from the FOS load catalog.

FOS User Interface:

- Load Catalog receives requests to generate loads from load contents files.
- Load Catalog receives constraint override status in response to soft constraints.
- Load Catalog sends a CMS status to indicate conflict information which was discovered in the constraint checking process.
- Load Catalog sends a CMS status to convey the success or failure of complete load generation.

FOS Planning and Scheduling:

- Load Catalog receives a request to update the count of scheduled uplinks for a particualr load.
- Load Catalog sends notification that a load has been deleted from the FOS load catalog.

CMS Schedule Controller:

- Load Catalog receives a request to delete a load or loads from the FOS load catalog.
- Load Catalog receives a request to store an ATC Load and create a catalog entry for that load.
- Load Catalog receives a query into the catalog to discover if a specified load has been uplinked or not.
- Load Catalog sends a status to indicate the success or failure of the above requests.

FOS Data Management:

- Load Catalog retrieves load contents files, which are used to generate loads.
- Load Catalog retrieves table formats, which are used to correctly format table data for table load generation.
- Load Catalog updates entries in the FOS load catalog.
- Load Catalog receives notification event messages that indicate tthe time at which a specified load has been successfully uplinked to the spacecraft.
- Load Catalog receives notification event messages whenever a table load contents file has been imported into DMS from the FDF.
- Load Catalog sends uplink loads, load reports, load images, and load contents files to be stored whenever such files are generated during the load generation process.
- Load Catalog sends a request to add an entry to the FOS load catalog.
- Load Catalog sends event messages.

CMS Command Model:

- Load Catalog sends RTS load contents files to be constraint checked.
- Load Catalog receives conflict information concerning constraint violations that were detected by the Rule-Based Constraint Checker.

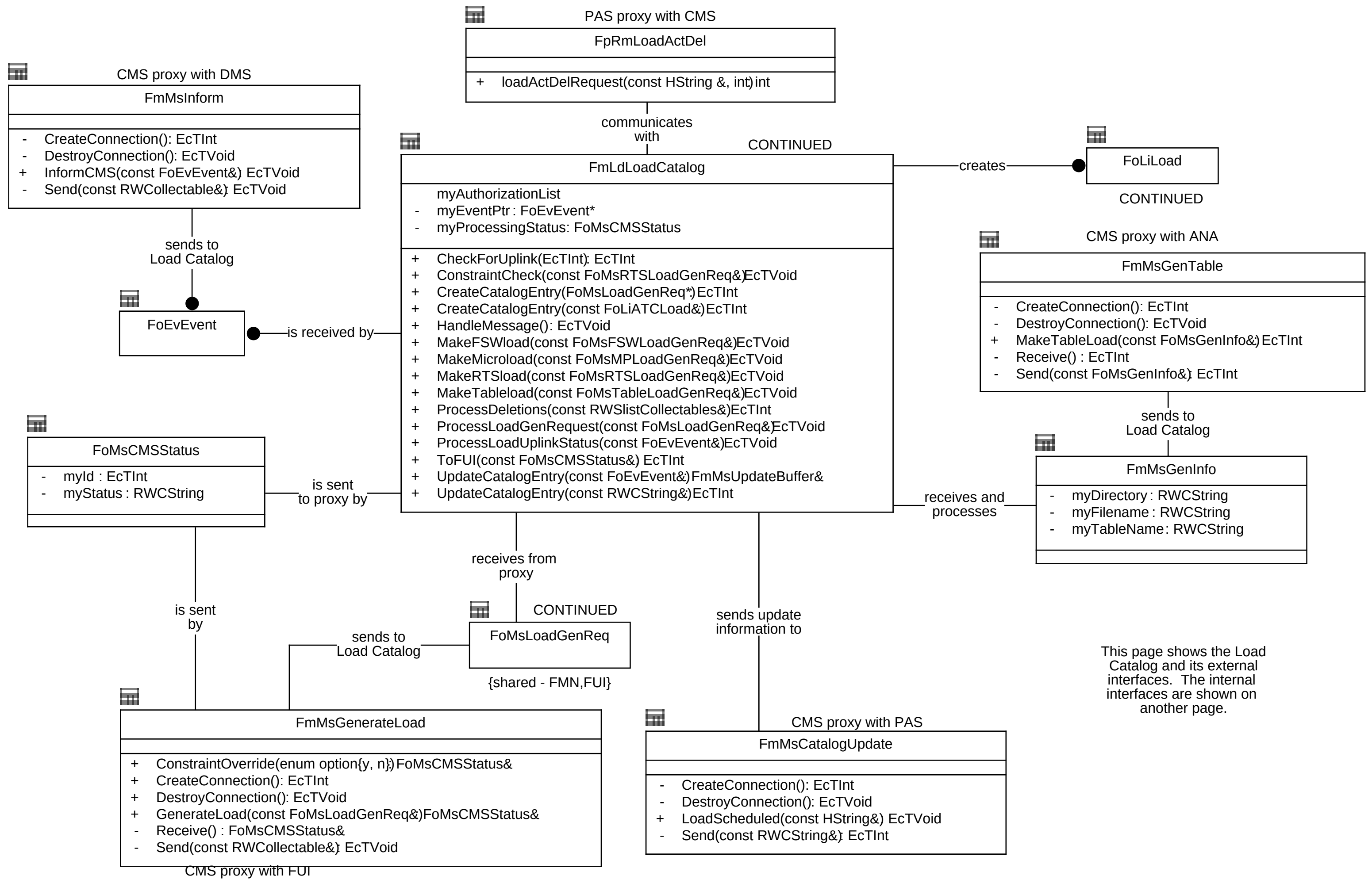
3.6.2 Load Catalog Interfaces

Table 3.6.2. Load Catalog Interfaces (1 of 2)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Generate Load	FmMsGenerateLoad	Proxy between CMS: Load Catalog and FUI.	CMS: Load Catalog	FUI: Load Manager, Table Load Builder, or RTS Load Builder	1/day
	FoMsLoadGenReq	Request to generate specified load.			
	FoMsCMSStatus	Status of load generation.			
Override RTS Constraints	FmMsGenerateLoad	Proxy between CMS: Load Catalog and FUI requesting that RTS constraint violations be ignored.	CMS: Load Catalog	FUI: RTS Load Builder	1/day
	FoMsCMSStatus	Status of constraint check.			
Update Load Scheduled Count	FmMsCatalogUpdate	Proxy between CMS: Load Catalog and PAS to update status of load to scheduled.	CMS:Load Catalog	PAS: Load Scheduler	1/day
Load Deletion	FpRmLoadActDel	Proxy between CMS: Load Catalog and PAS to request deletion of a load.	CMS: Load Catalog	PAS: Load Scheduler	
Process Load Uplink Notification	FmMsInform	Proxy between CMS: Load Catalog and DMS to request CMS act upon a load uplink event message.	CMS:Load Catalog	DMS: Event Handler	5/day
Check for ATC Load	FmMsStoreATCLoad	Proxy between CMS: Load Catalog and CMS: Schedule Controller to check for the existence of a load.	CMS:Load Catalog	CMS: Schedule Controller	1/week
ATC Load Deletion Request	FmMsStoreATCLoad	Proxy between CMS: Load Catalog and CMS: Schedule Comtroller to request deletion of a scheduled load.	CMS: Load Catalog	CMS: Schedule Controller	1/week
Store ATC Load	FmMsStoreATCLoad	Proxy between CMS: Load Catalog and CMS: Schedule Controller to request storage of an ATC load.	CMS: Load Catalog	CMS: Schedule Controller	5/day

Table 3.6.2. Load Catalog Interfaces (2 of 2)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Delete Buffer Models	FmSmMapBuffer	Proxy between CMS: Load Catalog and CMS: Spacecraft Model .	CMS: Spacecraft Model	CMS: Load Catalog	10/week
	FmMsDeleteATC Buffers	Request to delete buffers.			
Update Buffer Models	FmSmMapBuffer	Proxy between CMS: Load Catalog and CMS: Spacecraft Model.	CMS: Spacecraft Model	CMS: Load Catalog	10/week
	FmMsUpdateBuff er	Update Buffer from working to predicted and then to actual.			
Validate RTS	FmMsValidateCo nstraints	Proxy between CMS: Load Catalog and CMS: COmmand Model to request an RTS load contents be constraint checked.	CMS: Command Model	CMS: Load Catalog	1/day
	FoMsCMSStatus	Status of constraint check.			
Generate Table Load	FmMsGenTable	Proxy between CMS: Load Catalog and FAS.	CMS: Load Catalog	FAS: FaCcClockErr or	1/week
	FoMsGenInfo	Request to generate table load from load contents file.			
Store Load	FoDsFileAccessor	Proxy between DMS and CMS: Load Catalog to store files.	DMS	CMS: Load Catalog	50/day
Store Catalog Entry	FdDbAccessor	Proxy between DMS and CMS: Load Catalog to store catalog entries.	DMS	CMS: Load Catalog	5/day
Retrieve Catalog Entry	FdDbAccessor	Proxy between DMS and CMS: Load Catalog to retrieve catalog entries.	DMS	CMS: Load Catalog	5/day



PAS proxy with CMS
FpRmLoadActDel
+ loadActDelRequest(const HString &, int) int
communicates with
CONTINUED
CMS proxy with DMS
FmMsInform
- CreateConnection(): EcTInt
- DestroyConnection(): EcTVoid
+ InformCMS(const FoEvEvent&) EcTVoid
- Send(const RWCollectable&) EcTVoid
FmLdLoadCatalog
myAuthorizationList
- myEventPtr : FoEvEvent*
- myProcessingStatus: FoMsCMSStatus
+ CheckForUplink(EcTInt): EcTInt
+ ConstraintCheck(const FoMsRTSLoadGenReq&) EcTVoid
+ CreateCatalogEntry(FoMsLoadGenReq*) EcTInt
+ CreateCatalogEntry(const FoLiATCLoad&) EcTInt
+ HandleMessage(): EcTVoid
+ MakeFSWload(const FoMsFSWLoadGenReq&) EcTVoid
+ MakeMicroload(const FoMsMPLoadGenReq&) EcTVoid
+ MakeRTSload(const FoMsRTSLoadGenReq&) EcTVoid
+ MakeTableload(const FoMsTableLoadGenReq&) EcTVoid
+ ProcessDeletions(const RWSlistCollectables&) EcTInt
+ ProcessLoadGenRequest(const FoMsLoadGenReq&) EcTVoid
+ ProcessLoadUplinkStatus(const FoEvEvent&) EcTVoid
+ ToFUI(const FoMsCMSStatus&) EcTInt
+ UpdateCatalogEntry(const FoEvEvent&) FmMsUpdateBuffer&
+ UpdateCatalogEntry(const RWCString&) EcTInt
creates
FoLiLoad
CONTINUED
CMS proxy with ANA
FmMsGenTable
- CreateConnection(): EcTInt
- DestroyConnection(): EcTVoid
+ MakeTableLoad(const FoMsGenInfo&) EcTInt
- Receive() : EcTInt
- Send(const FoMsGenInfo&) EcTInt
sends to Load Catalog
FmMsGenInfo
- myDirectory : RWCString
- myFilename : RWCString
- myTableName : RWCString
receives and processes
sends to Load Catalog
FoEvEvent
is received by
FoMsCMSStatus
- myId : EcTInt
- myStatus : RWCString
is sent to proxy by
receives from proxy
CONTINUED
FoMsLoadGenReq
{shared - FMN,FUI}
is sent by
sends to Load Catalog
sends update information to
FmMsGenerateLoad
+ ConstraintOverride(enum option{y, n}) FoMsCMSStatus&
+ CreateConnection(): EcTInt
+ DestroyConnection(): EcTVoid
+ GenerateLoad(const FoMsLoadGenReq&) FoMsCMSStatus&
- Receive() : FoMsCMSStatus&
- Send(const RWCollectable&) EcTVoid
CMS proxy with FUI
CMS proxy with PAS
FmMsCatalogUpdate
- CreateConnection(): EcTInt
- DestroyConnection(): EcTVoid
+ LoadScheduled(const HString&) EcTVoid
- Send(const RWCString&) EcTInt
This page shows the Load Catalog and its external interfaces. The internal interfaces are shown on another page.

Figure 3.6-2. Load Catalog External Interfaces

3.6.3 Load Catalog Object Model

The Load Catalog object model is shown in Figure 3.6-2. FmLdLoadCatalog maintains lists, represented by FoLdCatalogEntry, of the various types of loads ready for uplink in the EOC. FmLdLoadCatalog is responsible for generating loads, creating and updating catalog entries, responding to queries about the load catalog, and generating reports.

The FmMsGenerateLoad class is an interface proxy class to FUI. FUI uses this class to request that a load be generated from a load contents file. FmLdLoadCatalog receives from this proxy a FoMsLoadGenReq object. This object contains information necessary to generate certain types of loads. The Load Catalog uses the FoMsLoadGenReq, shown in Figure 3.6-3, to find the load contents file and create the load. Also, some of the information in the FoMsLoadGenReq will be copied into the FoLdCatalogEntry. FmLdLoadCatalog returns to the proxy a FoMsCMSStatus object, which is used to convey the status of load generation.

The FmMsInform class is an interface proxy with DMS. DMS uses this class to notify FmLdLoadCatalog that either a load has been successfully uplinked, or that an externally-generated load contents file has been imported into DMS.

The FmMsGenTable class is an interface proxy to the Analysis subsystem. It is used when Analysis wants to request an FoLiTableLoad be generated from a load contents file. The class FoMsGenInfo contains information about the table load contents file from which Analysis would like a table load generated. FmLdLoadCatalog will use the data in FoMsGenInfo to generate the FoLiTableLoad.

The FmMsCatalogUpdate class is an interface proxy to the Planning and Scheduling subsystem. Planning and Scheduling uses this class to inform CMS that a load has been scheduled for uplink. FmLdLoadCatalog will update the FoLdCatalogEntry for that load appropriately.

The FpRmLoadActDel class is an interface proxy belonging to Planning and Scheduling that FmLdLoadCatalog uses to inform Planning and Scheduling when a load has been deleted from the FOS load catalog.

Figure 3.6-4 shows the different types of loads and their structures. Each of the load subclasses (FoLiTableLoad, FoLiFlightLoad, FoLiRTSLoad, FoLiMicroLoad, and FoLiATCLoad) derived from the FoLiLoad base class is responsible for building its load uplink module by packetizing the load data and appending the load initiate command. Additionally, the ATC and RTS loads contain commands which must be converted to binary to build the load uplink module, and certain fields in table loads must be converted to spacecraft table format. Additionally, the FoLiRTSLoad class validates commands in RTS loads using definitions in the Command Database.

The FoLiTableLoad class also uses the FoFmTableFormat class, which contains the template for the table load. This class is retrieved from DMS, and it contains one to many FoFmDataField objects.

Each FoLiLoad object is made up of four components. Figure 3.6-5 shows the components that make up each type of load. FoLiUplinkLoad represents the uplinkable, spacecraft-ready form of the load. FoLiLoadContents represents the raw contents of the load as they are received by CMS. FoLiLoadImage is the binary form of the load, before it has been put into spacecraft format. FoLiLoadReport is the report that is automatically generated whenever a load is generated. It contains pertinent information about the load and its location in spacecraft memory.

This page of the object shows different types of load generation requests that are received from FUI.

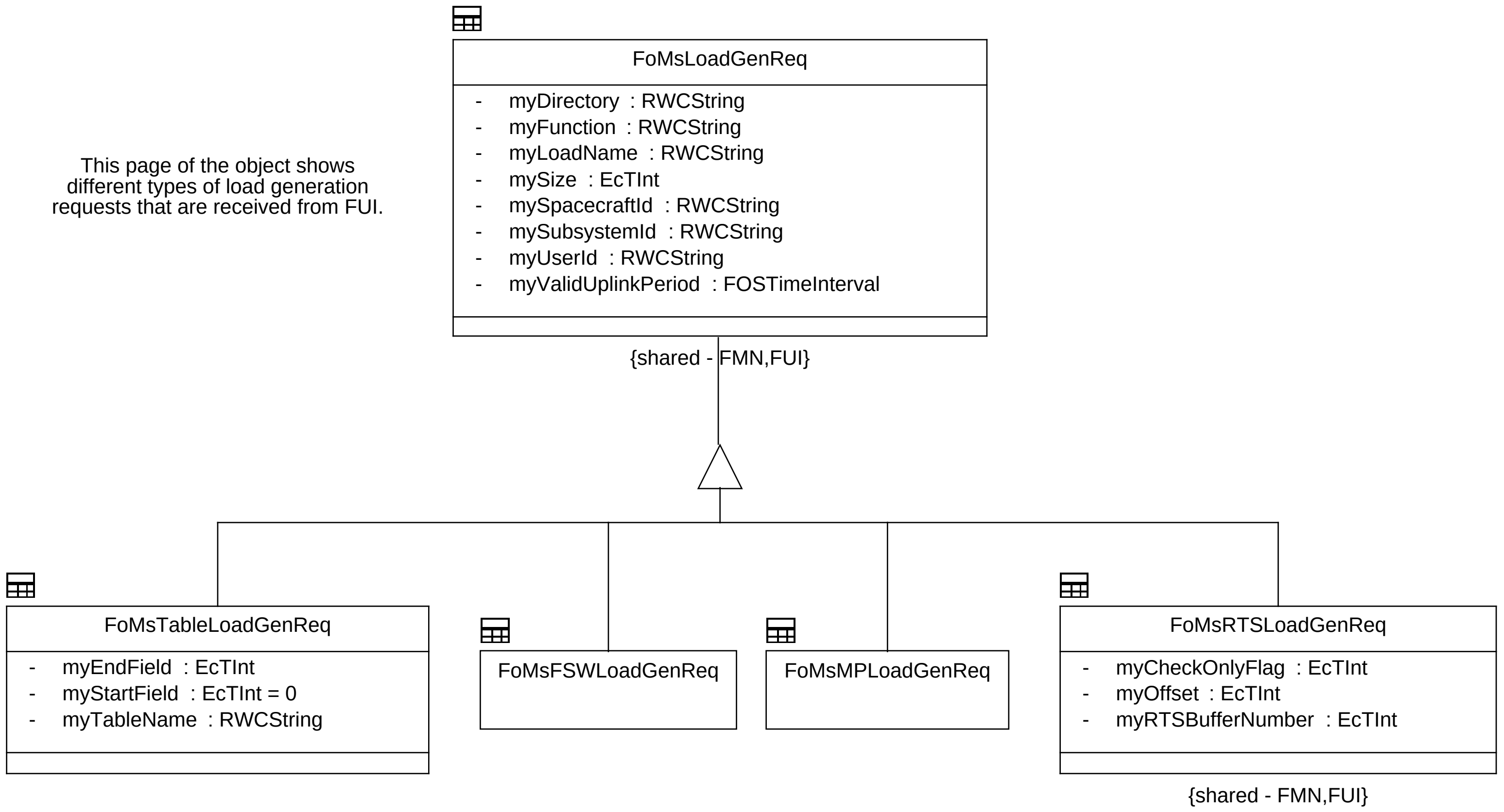


Figure 3.6-3. Load Generation Requests

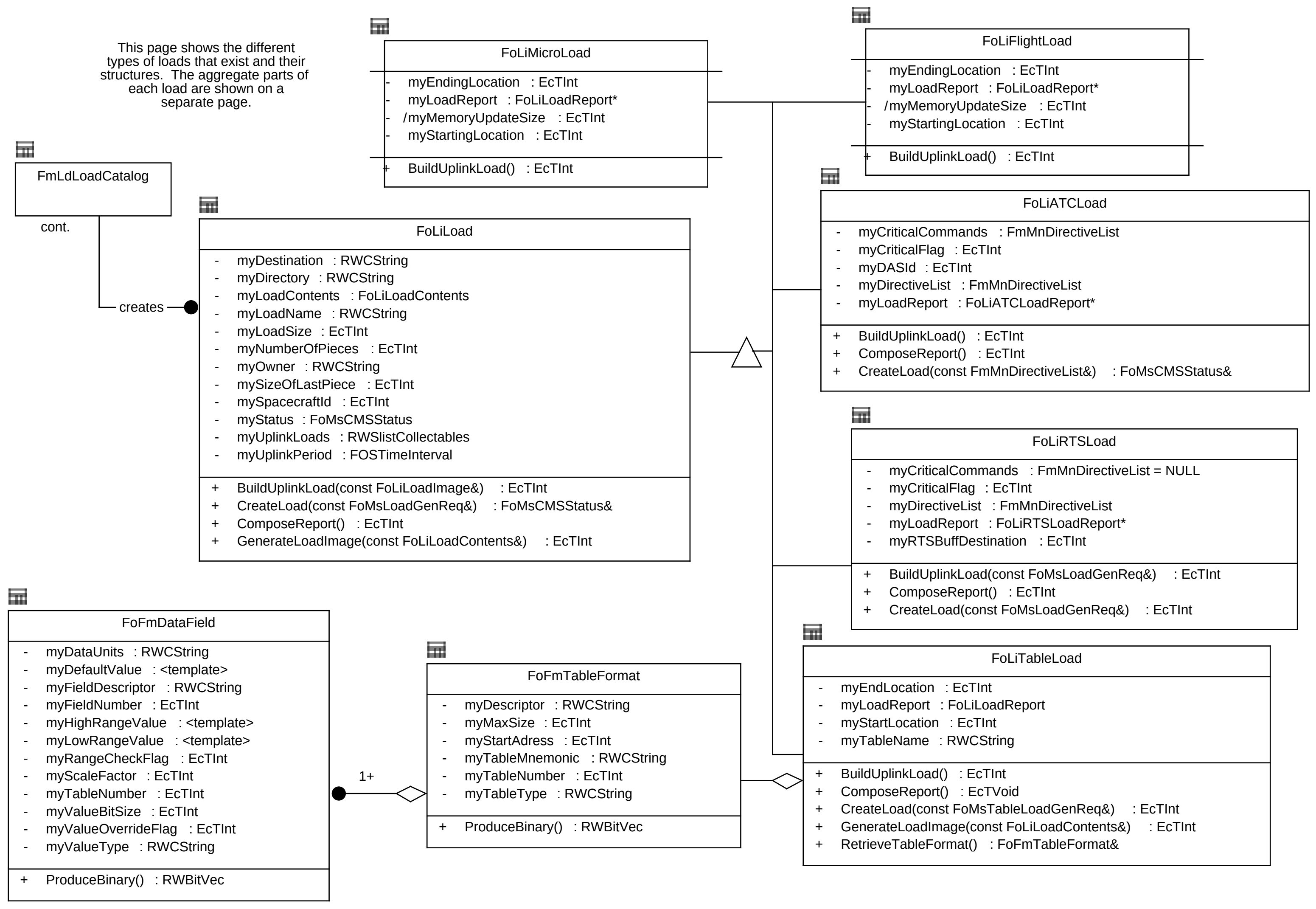
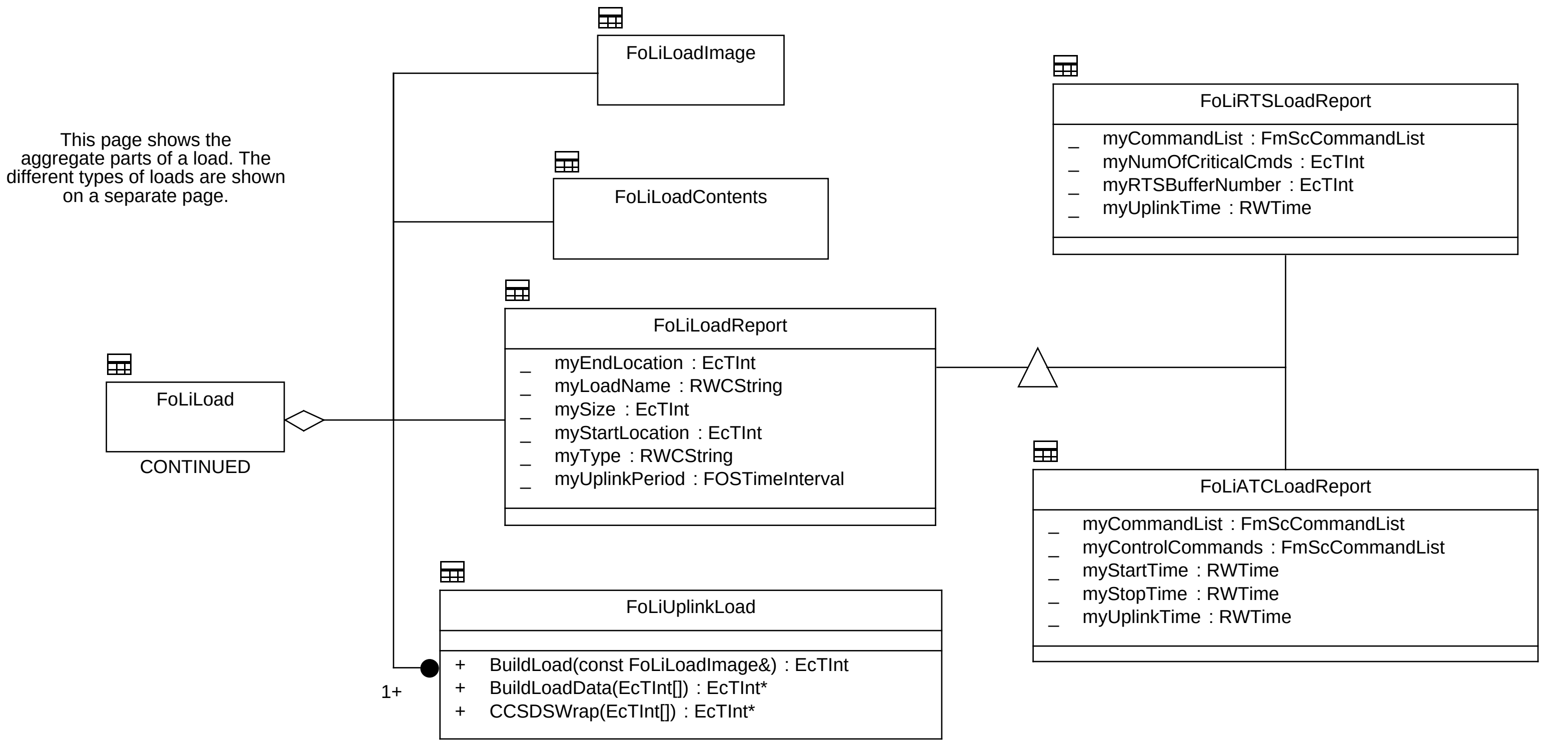

This page shows the different types of loads that exist and their structures. The aggregate parts of each load are shown on a separate page.
FmLdLoadCatalog
cont.
creates
FoLiMicroLoad
- myEndingLocation : EcTInt
- myLoadReport : FoLiLoadReport*
- /myMemoryUpdateSize : EcTInt
- myStartingLocation : EcTInt
+ BuildUplinkLoad() : EcTInt
FoLiFlightLoad
- myEndingLocation : EcTInt
- myLoadReport : FoLiLoadReport*
- /myMemoryUpdateSize : EcTInt
- myStartingLocation : EcTInt
+ BuildUplinkLoad() : EcTInt
FoLiLoad
- myDestination : RWCString
- myDirectory : RWCString
- myLoadContents : FoLiLoadContents
- myLoadName : RWCString
- myLoadSize : EcTInt
- myNumberOfPieces : EcTInt
- myOwner : RWCString
- mySizeOfLastPiece : EcTInt
- mySpacecraftId : EcTInt
- myStatus : FoMsCMSStatus
- myUplinkLoads : RWSlistCollectables
- myUplinkPeriod : FOSTimeInterval
+ BuildUplinkLoad(const FoLiLoadImage&) : EcTInt
+ CreateLoad(const FoMsLoadGenReq&) : FoMsCMSStatus&
+ ComposeReport() : EcTInt
+ GenerateLoadImage(const FoLiLoadContents&) : EcTInt
FoLiATCLoad
- myCriticalCommands : FmMnDirectiveList
- myCriticalFlag : EcTInt
- myDASId : EcTInt
- myDirectiveList : FmMnDirectiveList
- myLoadReport : FoLiATCLoadReport*
+ BuildUplinkLoad() : EcTInt
+ ComposeReport() : EcTInt
+ CreateLoad(const FmMnDirectiveList&) : FoMsCMSStatus&
FoLiRTSLoad
- myCriticalCommands : FmMnDirectiveList = NULL
- myCriticalFlag : EcTInt
- myDirectiveList : FmMnDirectiveList
- myLoadReport : FoLiRTSLoadReport*
- myRTSBuffDestination : EcTInt
+ BuildUplinkLoad(const FoMsLoadGenReq&) : EcTInt
+ ComposeReport() : EcTInt
+ CreateLoad(const FoMsLoadGenReq&) : EcTInt
FoFmDataField
- myDataUnits : RWCString
- myDefaultValue : <template>
- myFieldDescriptor : RWCString
- myFieldNumber : EcTInt
- myHighRangeValue : <template>
- myLowRangeValue : <template>
- myRangeCheckFlag : EcTInt
- myScaleFactor : EcTInt
- myTableNumber : EcTInt
- myValueBitSize : EcTInt
- myValueOverrideFlag : EcTInt
- myValueType : RWCString
+ ProduceBinary() : RWBitVec
1+
FoFmTableFormat
- myDescriptor : RWCString
- myMaxSize : EcTInt
- myStartAdress : EcTInt
- myTableMnemonic : RWCString
- myTableNumber : EcTInt
- myTableType : RWCString
+ ProduceBinary() : RWBitVec
FoLiTableLoad
- myEndLocation : EcTInt
- myLoadReport : FoLiLoadReport
- myStartLocation : EcTInt
- myTableName : RWCString
+ BuildUplinkLoad() : EcTInt
+ ComposeReport() : EcTVoid
+ CreateLoad(const FoMsTableLoadGenReq&) : EcTInt
+ GenerateLoadImage(const FoLiLoadContents&) : EcTInt
+ RetrieveTableFormat() : FoFmTableFormat&

Figure 3.6-4. Load Types


This page shows the aggregate parts of a load. The different types of loads are shown on a separate page.
FoLiLoad
CONTINUED
FoLiLoadImage
FoLiLoadContents
FoLiLoadReport
_ myEndLocation : EcTInt
_ myLoadName : RWCString
_ mySize : EcTInt
_ myStartLocation : EcTInt
_ myType : RWCString
_ myUplinkPeriod : FOSTimeInterval
FoLiUplinkLoad
+ BuildLoad(const FoLiLoadImage&) : EcTInt
+ BuildLoadData(EcTInt[]) : EcTInt*
+ CCSDSWrap(EcTInt[]) : EcTInt*
1+
FoLiRTSLoadReport
_ myCommandList : FmScCommandList
_ myNumOfCriticalCmds : EcTInt
_ myRTSBufferNumber : EcTInt
_ myUplinkTime : RWTime
FoLiATCLoadReport
_ myCommandList : FmScCommandList
_ myControlCommands : FmScCommandList
_ myStartTime : RWTime
_ myStopTime : RWTime
_ myUplinkTime : RWTime

3.6-5. Load Components

Derived from this class is FoLiRTSLoadReport, which contains additional information relevant to RTS loads only, such as a buffer number and number of commands. Also derived from FoLiLoadReport is FoLiATCLoadReport, which contains information pertinent only to ATC loads, such as a listing of commands and buffer information. Figure 3.6-6 shows that these components are all derived from FoDsFile.

Figure 3.6-7 shows the internal interfaces of FmLdLoadCatalog. The constraint checking performed on RTS loads is done through FmMsValidateConstraints, the interface proxy to the Command Model. The proxy will accept the request for constraint checking to be done on a specified RTS load contents, and return an FoMsCMSStatus object, indicating what constraint violations were found, if any.

The class FmSmMapBuffer is a proxy to the spacecraft model. It is used to request updating of the spacecraft model when an FoEvEvent specifying a successful load uplink is received. FmLdLoadCatalog sends a FoMsUpdateBuffer object via the proxy. The proxy is also used to request deletion of any ATC buffers in the event of a late change. The FoEvEvent specifying the load uplink is received through FmMsInform, a proxy with the Data Management Subsystem.

The class FmMsStoreATCLoad is a proxy to the FmScScheduleController. It is used to receive a reference to an FoLiATCLoad object which was created by FmScScheduleController. The FoLiATCLoad is stored in DMS, and a FoLdCatalogEntry is created for it and stored as well. This proxy can also request Load Catalog to verify that a certain load has or has not been uplinked.

3.6.4 Load Catalog Dynamic Model

The Load Catalog Dynamic Model described in this section consists of the following scenarios:

- Load Catalog Initialization
- Table Load Generation
- Table Load Generation from FDF Load Contents
- Table Load Generation for Clock Correlation
- RTS Load Generation
- Microprocessor Load Generation
- Flight Software Load Generation
- ATC Load Generation
- Uplink Notification Receipt

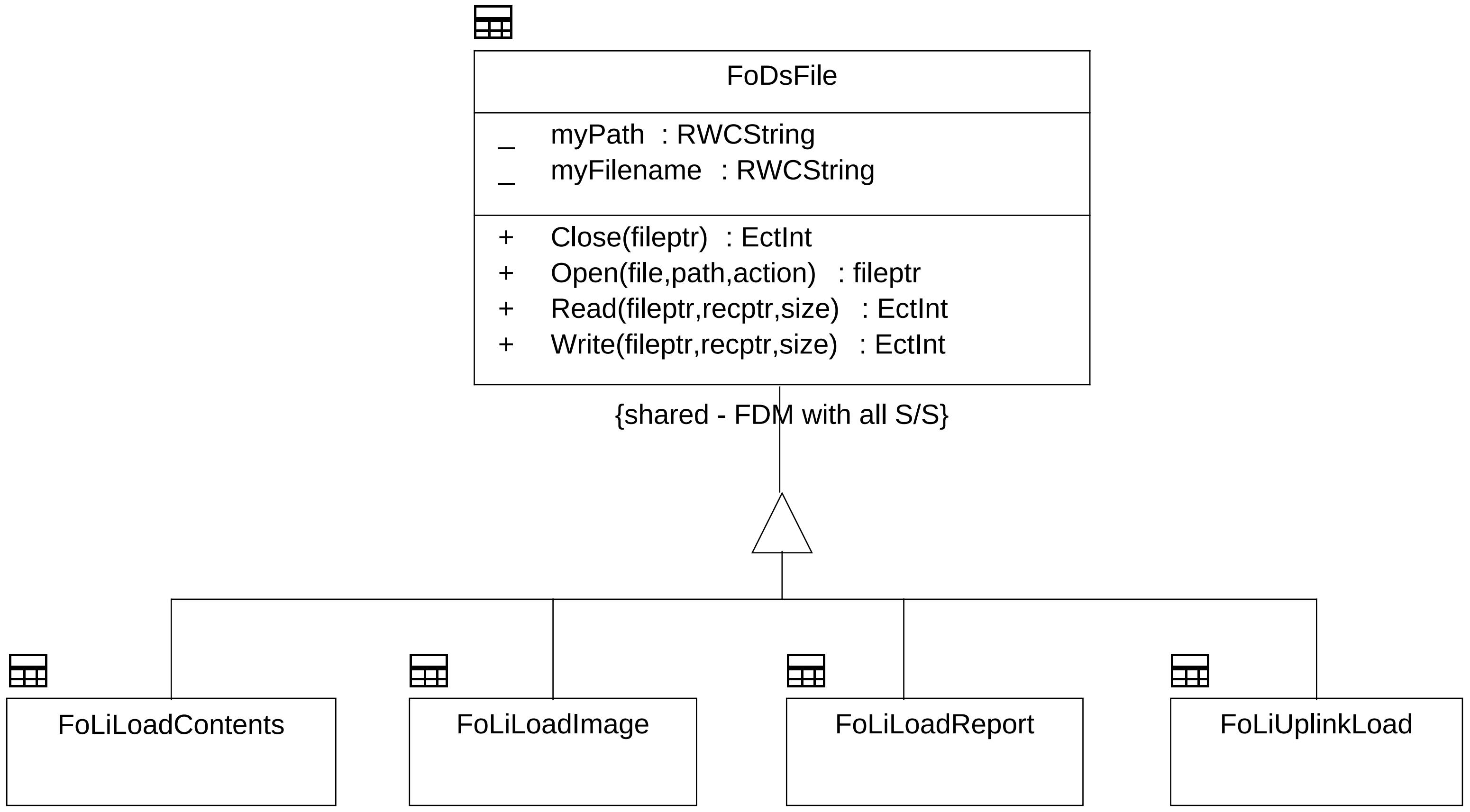
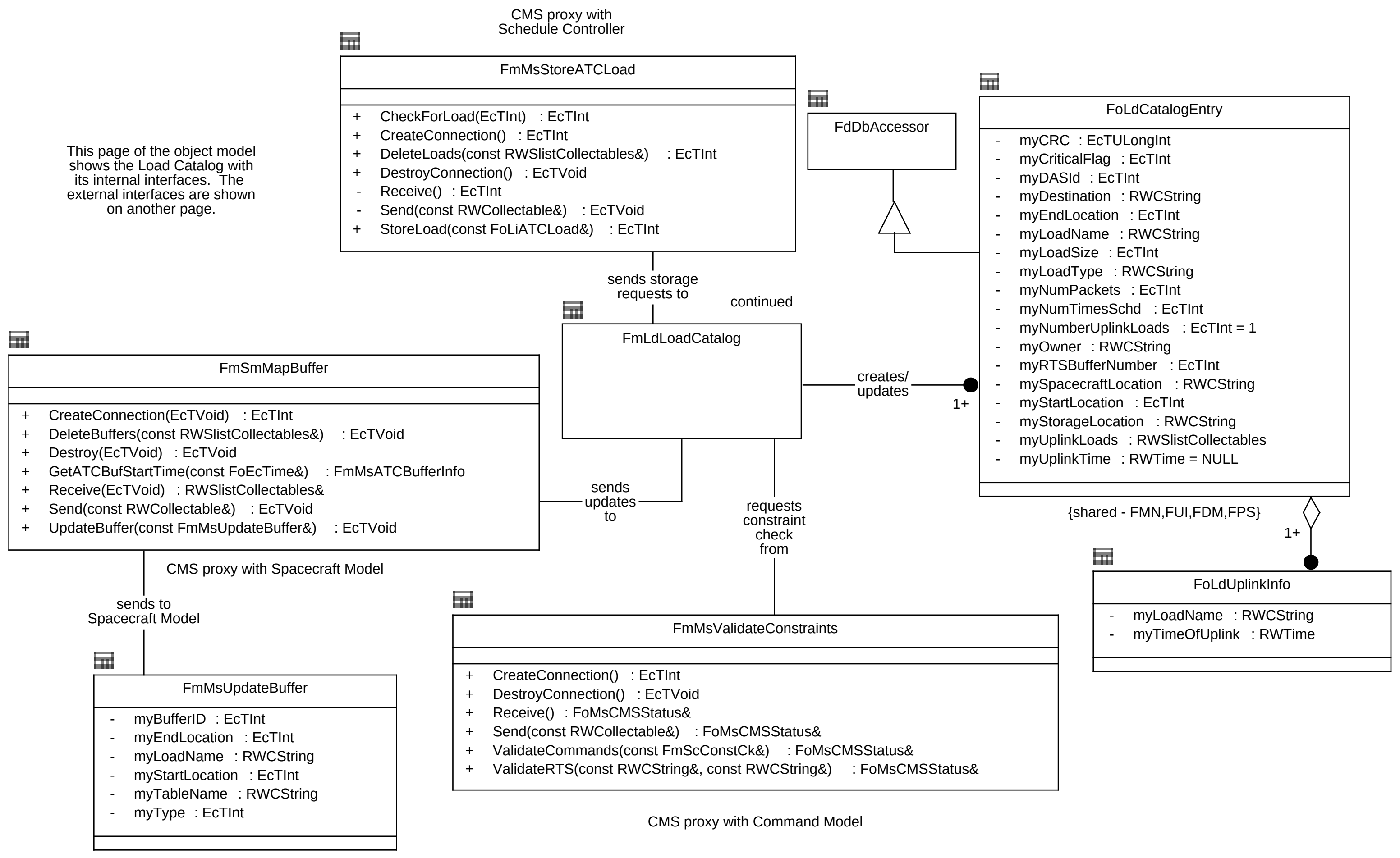

FoDsFile
_ myPath : RWCString
_ myFilename : RWCString
+ Close(fileptr) : EctInt
+ Open(file,path,action) : fileptr
+ Read(fileptr,recptr,size) : EctInt
+ Write(fileptr,recptr,size) : EctInt
{shared - FDM with all S/S}
FoLiLoadContents
FoLiLoadImage
FoLiLoadReport
FoLiUplinkLoad

Figure 3.6-6. Load Component File Classes



CMS proxy with Schedule Controller
FmMsStoreATCLoad
+ CheckForLoad(EcTInt) : EcTInt
+ CreateConnection() : EcTInt
+ DeleteLoads(const RWSlistCollectables&) : EcTInt
+ DestroyConnection() : EcTVoid
- Receive() : EcTInt
- Send(const RWCollectable&) : EcTVoid
+ StoreLoad(const FoLiATCLoad&) : EcTInt
FdDbAccessor
FoLdCatalogEntry
- myCRC : EcTULongInt
- myCriticalFlag : EcTInt
- myDASId : EcTInt
- myDestination : RWCString
- myEndLocation : EcTInt
- myLoadName : RWCString
- myLoadSize : EcTInt
- myLoadType : RWCString
- myNumPackets : EcTInt
- myNumTimesSchd : EcTInt
- myNumberUplinkLoads : EcTInt = 1
- myOwner : RWCString
- myRTSBufferNumber : EcTInt
- mySpacecraftLocation : RWCString
- myStartLocation : EcTInt
- myStorageLocation : RWCString
- myUplinkLoads : RWSlistCollectables
- myUplinkTime : RWTime = NULL
This page of the object model shows the Load Catalog with its internal interfaces. The external interfaces are shown on another page.
sends storage requests to
continued
FmLdLoadCatalog
creates/ updates
1+
FmSmMapBuffer
+ CreateConnection(EcTVoid) : EcTInt
+ DeleteBuffers(const RWSlistCollectables&) : EcTVoid
+ Destroy(EcTVoid) : EcTVoid
+ GetATCBufStartTime(const FoEcTime&) : FmMsATCBufferInfo
+ Receive(EcTVoid) : RWSlistCollectables&
+ Send(const RWCollectable&) : EcTVoid
+ UpdateBuffer(const FmMsUpdateBuffer&) : EcTVoid
sends updates to
requests constraint check from
{shared - FMN,FUI,FDM,FPS}
1+
CMS proxy with Spacecraft Model
sends to Spacecraft Model
FoLdUplinkInfo
- myLoadName : RWCString
- myTimeOfUplink : RWTime
FmMsValidateConstraints
+ CreateConnection() : EcTInt
+ DestroyConnection() : EcTVoid
+ Receive() : FoMsCMSStatus&
+ Send(const RWCollectable&) : FoMsCMSStatus&
+ ValidateCommands(const FmScConstCk&) : FoMsCMSStatus&
+ ValidateRTS(const RWCString&, const RWCString&) : FoMsCMSStatus&
FmMsUpdateBuffer
- myBufferID : EcTInt
- myEndLocation : EcTInt
- myLoadName : RWCString
- myStartLocation : EcTInt
- myTableName : RWCString
- myType : EcTInt
CMS proxy with Command Model

Figure 3.6-7. Load Catalog Internal Interfaces

3.6.4.1 Load Catalog Initialization Scenario

3.6.4.1.1 Load Catalog Initialization Abstract

The Load Catalog Initialization scenario describes how the Load Catalog software gets started and readied for data processing.

3.6.4.1.2 Load Catalog Initialization Summary Information

Interfaces:

- Command Model
- Spacecraft Model
- Planning and Scheduling

Stimulation:

- Load Catalog software started and initialize function called

Desired Response:

- Load Catalog software ready to accept data for processing

Pre-Conditions:

- none

Post-Conditions:

- none

3.6.4.1.3Load Catalog Initialization Description

Figure 3.6-8 shows the Load Catalog Initialization Event Trace. When the initialize function is called, Load Catalog creates an ipc connection with the Command Model via the proxy FmMsValidateConstraints. Load catalog then creates an ipc connection to the Spacecraft Model via the proxy FmMsMapBuffer. Lastly, Load Catalog creates an ipc connection with Planning and Scheduling via the proxy FpRmLoadActDel. If all of these connections are successfully established, the initialize function is complete.

3.6.4.2 Table Load Generation Scenario

3.6.4.2.1 Table Load Generation Abstract

The Table Load Generation scenario describes the generation of an uplink table load from a table load contents file that was created by the user through the Table Editor tool provided by User Interface.

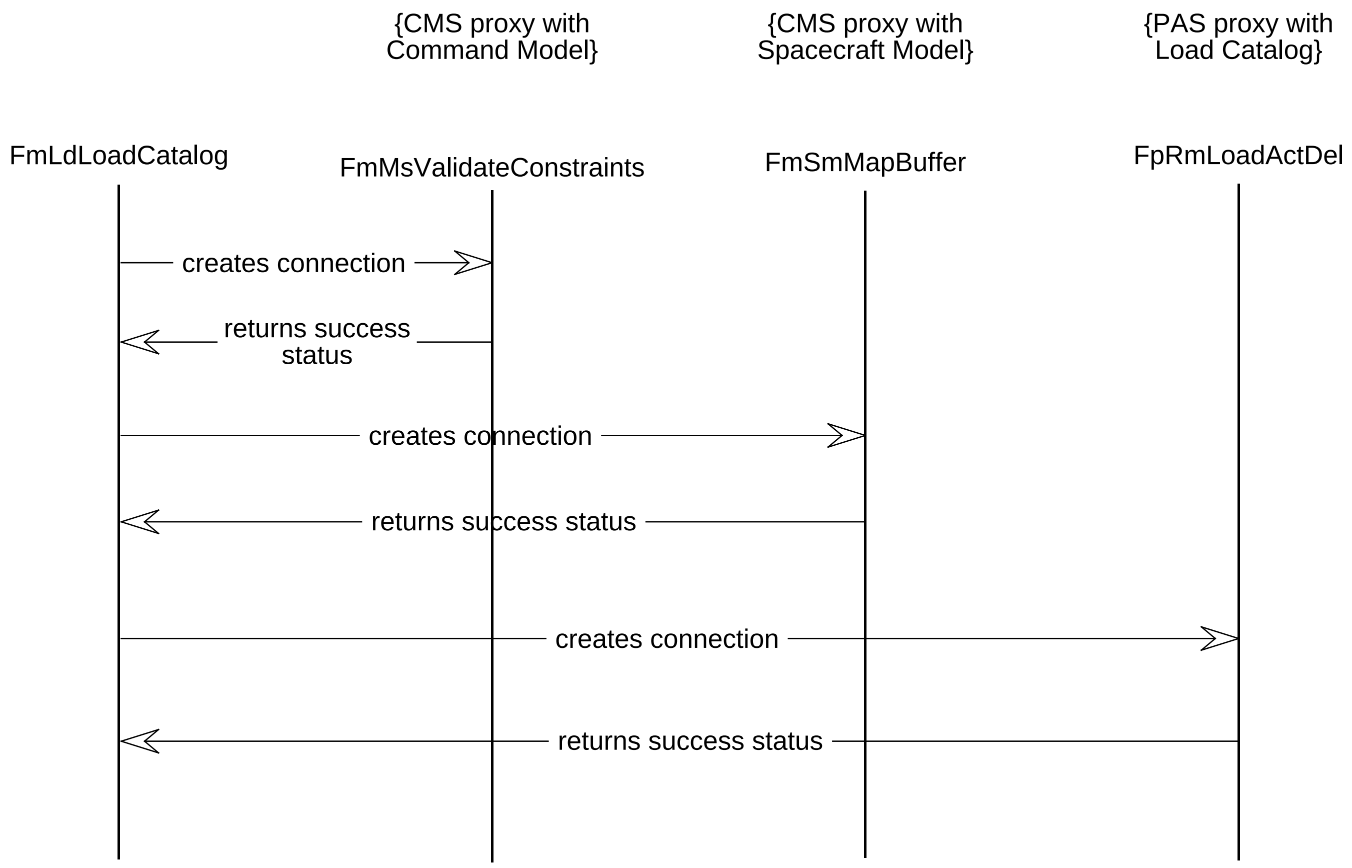


Figure 3.6-8. Load Catalog Initialization Event Trace

3.6.4.2.2Table Load Generation Summary Information

Interfaces:

- User Interface
- Data Management

Stimulus:

- Receipt of a Load Generation Request from FUI

Desired Response:

- Table Load generated and stored with DMS
- Table Load entered into Load Catalog

Pre-Conditions:

- Load Catalog software has been initiated

Post-Conditions:

- Generated load is available for uplink

3.6.4.2.3Table Load Generation Description

Figure 3.6-9 shows the Table Load Generation Event Trace. The Load Catalog receives a Load Generation Request, including identifying information about the load contents file, from FUI. Load Catalog creates a Table Load object and passes it the identifying information about the table load contents. Table Load retrieves the load contents from DMS. Table Load retrieves the table format from DMS, and uses it with the load contents to create the load image. Table Load generates the uplink table loads from the load image and requests DMS to store the uplink load, the load image,the load contents, and the load report. Load Catalog creates a Load Catalog Entry and requests DMS to add it to the load catalog database. Load Catalog sends an Event Message to DMS indicating that the load generation is complete and passes a success status to FUI.

3.6.4.3 Table Load Generation from FDF Load Contents Scenario

3.6.4.3.1Table Load Generation from FDF Load Contents Abstract

The Table Load Generation from FDF Load Contents scenario describes how a Table Load will be generated from externally generated load contents. When an FDF Table Load Contents file is imported into DMS, CMS is notified. Upon notification, CMS will generate a Table Load from the Table Load Contents and creates a Catalog Entry for the Table Load.

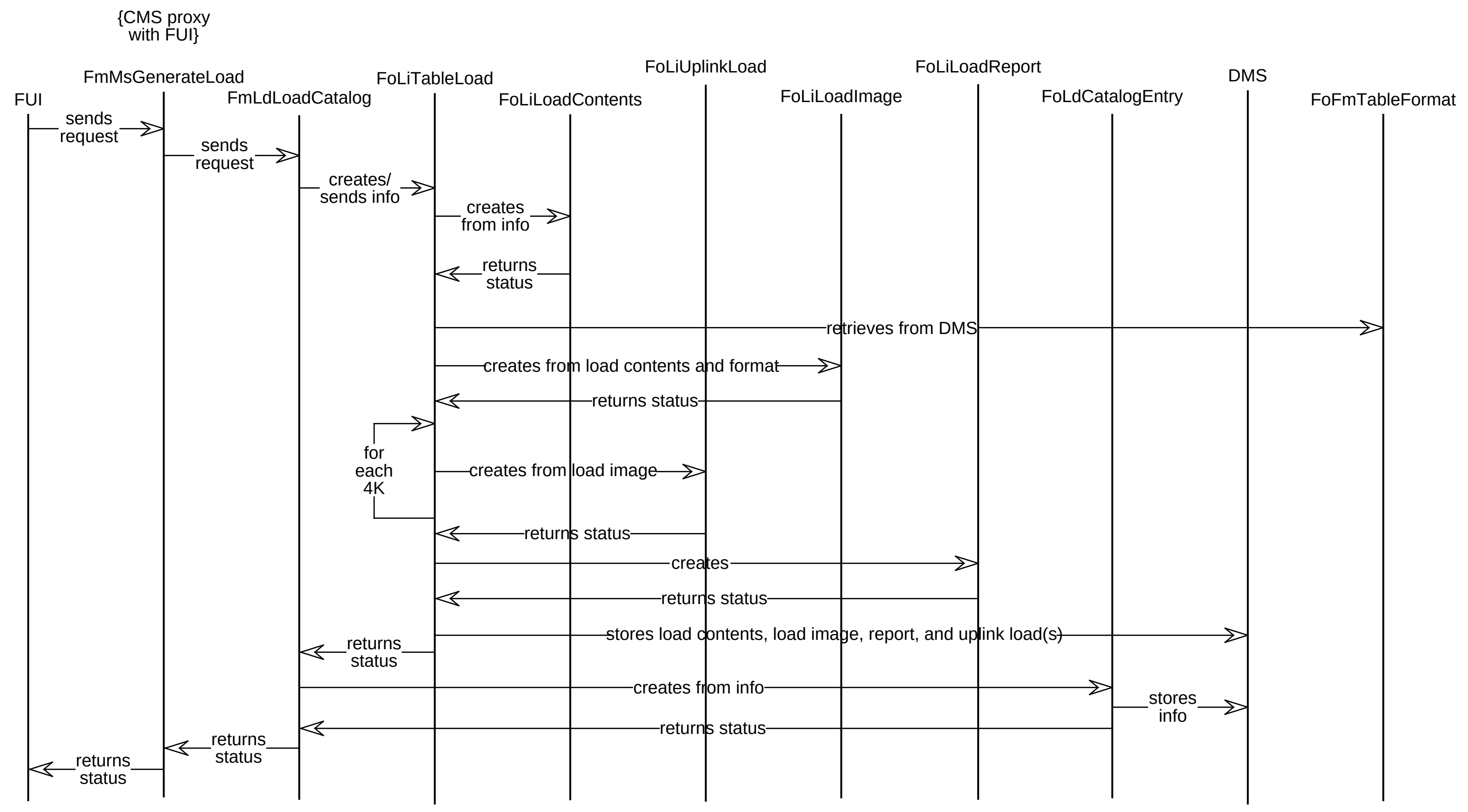


Figure 3.6-9. Table Load Generation Event Trace

3.6.4.3.2 Table Load Generation from FDF Load Contents Summary Information

Interfaces:

- DMS

Stimulus:

- Receipt of a FDF Load Receipt Notification Event Message from DMS

Desired Response:

- Table Load generated and stored with DMS
- Table Load entered into Load Catalog

Pre-Conditions:

- Load Catalog software has been initialized

Post-Conditions:

- Generated load is available for uplink

3.6.4.3.3Table Load Generation from FDF Load Contents Description

Figure 3.6-10 shows the Table Load Generation from FDF Load Contents Event Trace. DMS receives an externally-generated Load Contents file from the FDF. Once the load contents has been validated, DMS informs Load Catalog of the existence of the Load Contents file.

The Load Catalog creates a Table Load object and passes it the identifying information about the table load contents. Table Load retrieves the load contents from DMS. Table Load retrieves the table format from DMS, and uses it with the load contents to create the load image. Table Load generates the uplink table load from the load image, and then creates the load report. Load Catalog requests DMS to store the uplink load, the load image, the load contents, and the load report. Load Catalog creates a Load Catalog Entry and requests DMS to add it to the load catalog database. Load Catalog sends an Event Message to DMS indicating that the load generation is complete.

3.6.4.4 Table Load Generation for Clock Correlation Scenario

3.6.4.4.1 Table Load Generation for Clock Correlation Abstract

The Table Load Generation for Clock Correlation scenario describes how a Table Load is generated from a load contents file received from the Analysis subsystem.

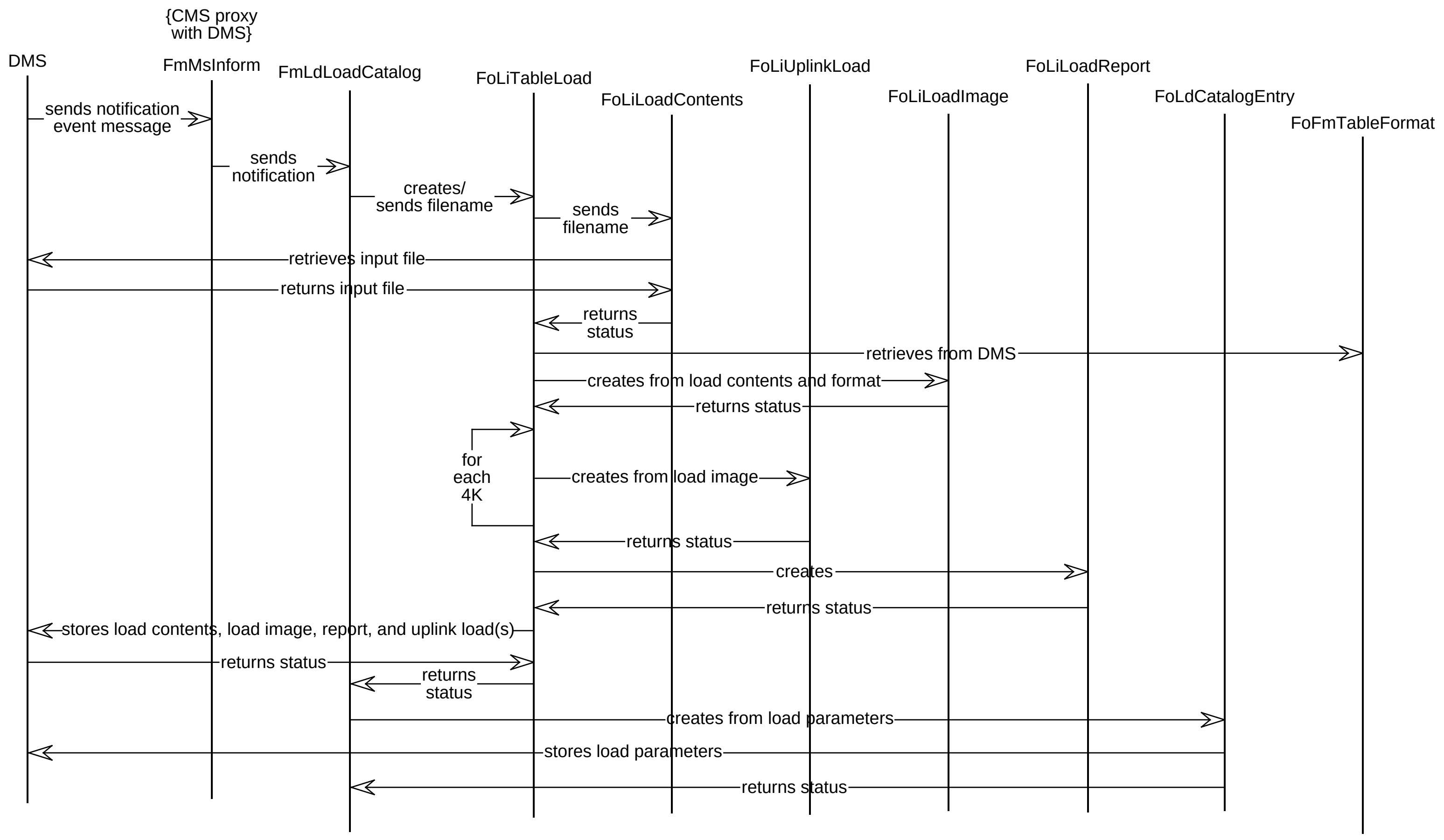


Figure 3.6-10. Table Load Generation from FDF Load Contents Event Trace

3.6.4.4.2 Table Load Generation for Clock Correlation Summary Information

Interfaces:

- Analysis
- DMS

Stimulus:

- Receipt of a table generation request from Analysis

Desired Response:

- Table Load generated and stored with DMS
- Table Load entered into Load Catalog

Pre-Conditions:

- Load Catalog software has been initiated

Post-Conditions:

- Table Load available for uplink

3.6.4.4.3 Table Load Generation for Clock Correlation Description

Figure 3.6-11 shows the Table Load Generation for Clock Correlation Event Trace. The Load Catalog receives a table load generation request from Analysis via the proxy FmMsGenTable. The Load Catalog reads the table load generation request to learn the location of the load contents file.

The Load Catalog creates a Table Load object and passes it the identifying information about the table load contents. Table Load retrieves the load contents from DMS. Table Load retrieves the table format from DMS, and uses it with the load contents to create the load image. Table Load generates the uplink table load from the load image, and then creates the load report. Load Catalog requests DMS to store the uplink load, the load image, the load contents, and the load report. Load Catalog creates a Load Catalog Entry and requests DMS to add it to the load catalog database. Load Catalog sends an Event Message to DMS indicating that the load generation is complete.

3.6.4.5 RTS Load Generation Scenario

3.6.4.5.1 RTS Load Generation Abstract

The RTS Load Generation scenario describes the generation of an uplink RTS load from an RTS load contents file that was created by the user through the RTSEditor tool provided by User Interface.

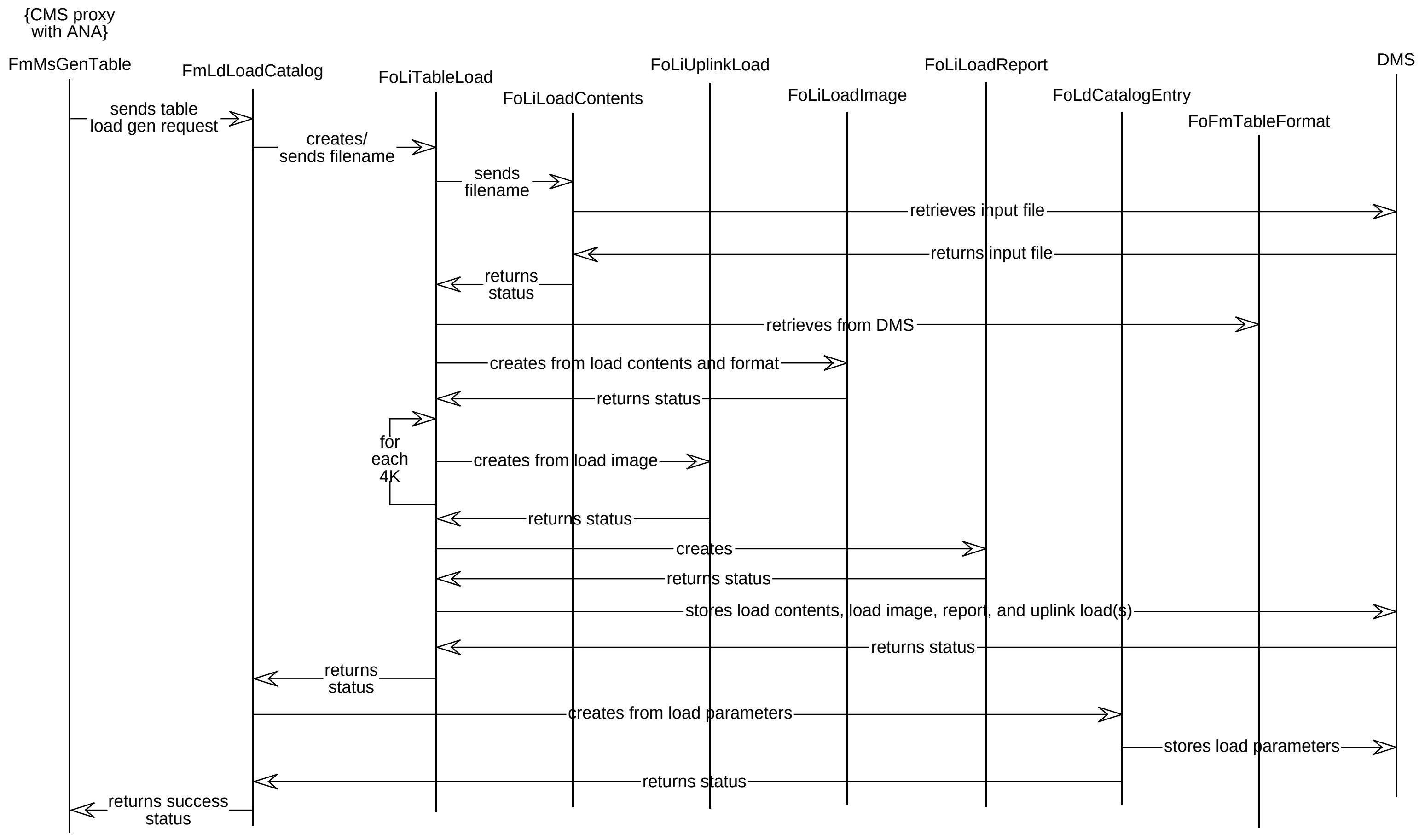


Figure 3.6-11. Table Load Generation for Clock Correlation Event Trace

3.6.4.5.2 RTS Load Generation Summary Information

Interfaces:

- User Interface
- Data Management
- Command Model

Stimulus:

- Receipt of Load Generation Request from FUI

Desired Response:

- RTS Load generated and stored with DMS
- RTS Load entered in RTS Load Catalog

Pre-Conditions:

- Load Catalog software has been initiated

Post-Conditions:

- Generated load is available for uplink

3.6.4.5.3 RTS Load Generation Description

Figure 3.6-12 shows the RTS Load Generation Event Trace. The Load Catalog receives a Load Generation Request, including identifying information about the load contents file, from FUI. Load Catalog requests a constraint check from the Command Model on the load contents file. Command Model returns a status, which is returned to FUI if constraints are found. FUI has the option to override soft constraints or terminate the load generation process. If no constraints violations are found, or if soft constraint violations are overridden, Load Catalog creates an RTS Load object and passes it the identifying information about the RTS load contents. RTS Load retrieves the load contents from DMS. RTS Load generates the uplink RTS load from the load contents and requests DMS to store the uplink load, load image, load contents, and the load report. Load Catalog creates a Load Catalog Entry and requests DMS to add it to the load catalog database. Load Catalog sends an Event Message to DMS indicating that the load generation is complete and passes a success status to FUI.

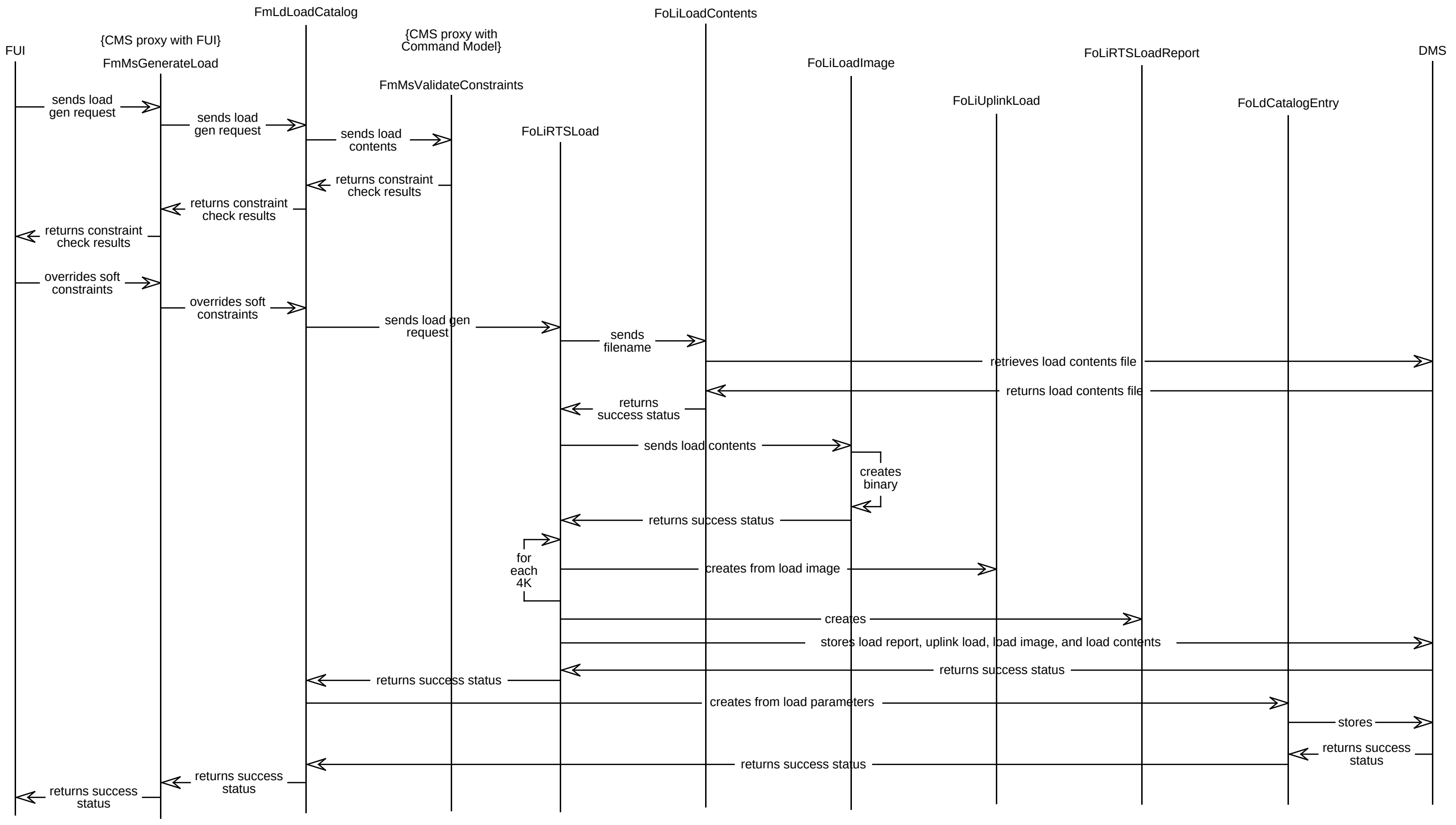


Figure 3.6-12. RTS Load Generation Event Trace

3.6.4.6 Microprocessor Load Generation Scenario

3.6.4.6.1 Microprocessor Load Generation Abstract

The Microprocessor Load Generation scenario describes the generation of an uplink microprocessor load from a microprocessor load contents file that was created outside of the FOS and imported via an IST or EOC workstation.

3.6.4.6.2 Microprocessor Load Generation Summary Information

Interfaces:

- User Interface
- Data Management

Stimulus:

- Receipt of Load Generation Request from FUI

Desired Response:

- Microprocessor Load generated and stored with DMS
- Microprocessor Load entered in microprocessor Load Catalog

Pre-Conditions:

- Load Catalog software has been initiated

Post-Conditions:

- Generated load is available for uplink

3.6.4.6.3 Microprocessor Load Generation Description

Figure 3.6-13 shows the Microprocessor Load Generation Event Trace. The Load Catalog receives a Load Generation Request, including identifying information about the load contents file, from FUI. Load Catalog creates a Microprocessor Load object and passes it the identifying information about the microprocessor load contents. Microprocessor Load retrieves the load contents from DMS. Microprocessor Load generates the uplink microprocessor load from the load contents and requests DMS to store the uplink load, load contents, load image, and the load report. Load Catalog creates a Load Catalog Entry and requests DMS to add it to the load catalog database. Load Catalog sends an Event Message to DMS indicating that the load generation is complete and passes a success status to FUI.

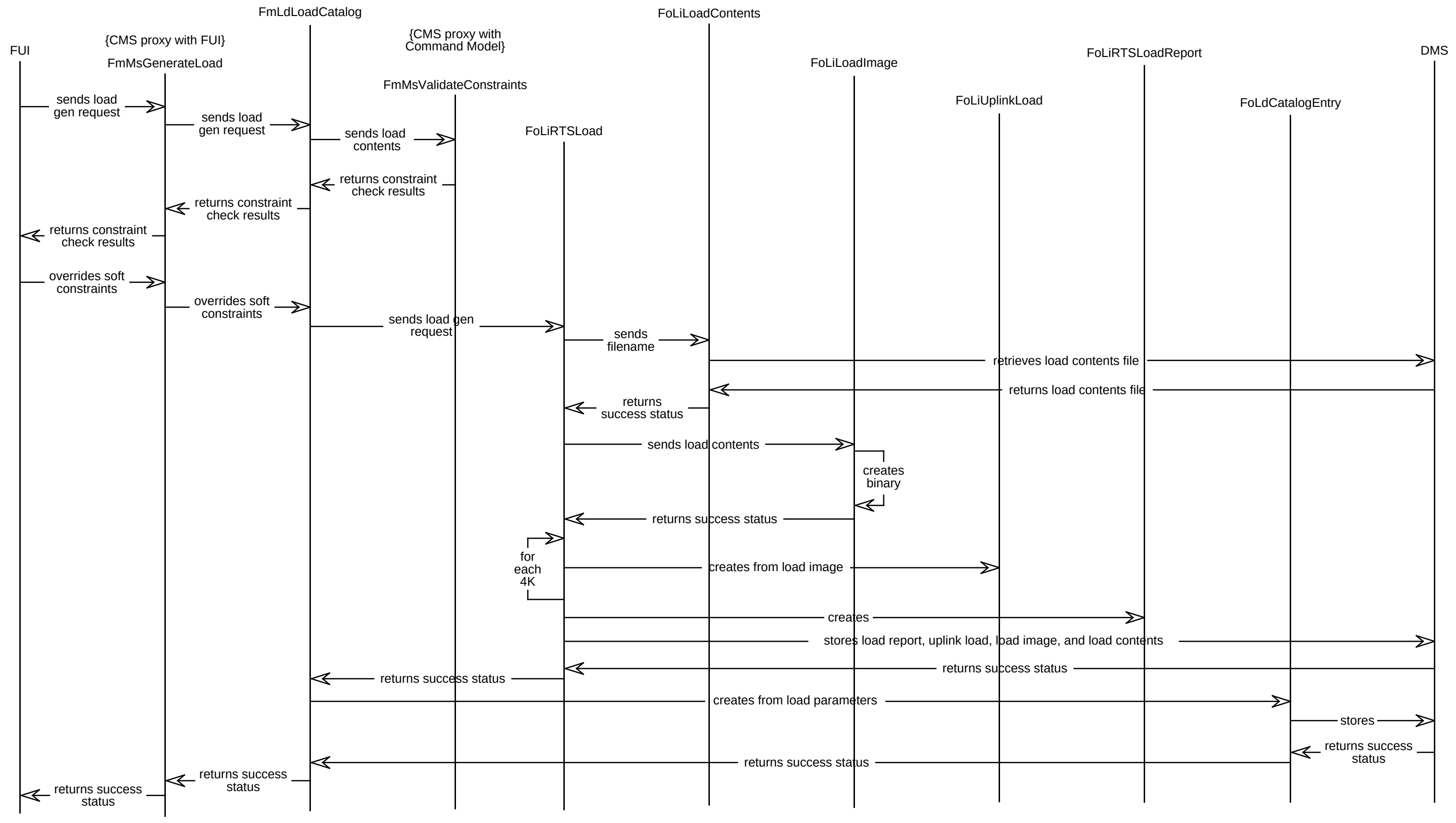


Figure 3.6-13. Microprocessor Load Generation Event Trace

3.6.4.7 Flight Software Load Generation Scenario

3.6.4.7.1 Flight Software Load Generation Abstract

The Flight Software Load Generation scenario describes the generation of an uplink flight software load from a flight software load contents file that was created outside of the FOS and imported via an IST or EOC workstation.

3.6.4.7.2 Flight Software Load Generation Summary Information

Interfaces:

- User Interface
- Data Management

Stimulus:

- Receipt of Load Generation Request from FUI

Desired Response:

- Flight Software Load generated and stored with DMS
- Flight Software Load entered in Flight software Load Catalog

Pre-Conditions:

- Load Catalog software has been initiated

Post-Conditions:

- Generated load is available for uplink

3.6.4.7.3 Flight Software Load Generation Description

Figure 3.6-14 shows the Flight Software Load Generaton Event Trace. The Load Catalog receives a Load Generation Request, including identifying information about the load contents file, from FUI. Load Catalog creates a Flight Software Load object and passes it the identifying information about the flight software load contents. Flight Software Load retrieves the load contents from DMS. Flight Software Load generates the uplink flight software load from the load contents and requests DMS to store the uplink load, load contents, load image, and the load report. Load Catalog creates a Load Catalog Entry and requests DMS to add it to the load catalog database. Load Catalog sends an Event Message to DMS indicating that the load generation is complete and passes a success status to FUI.

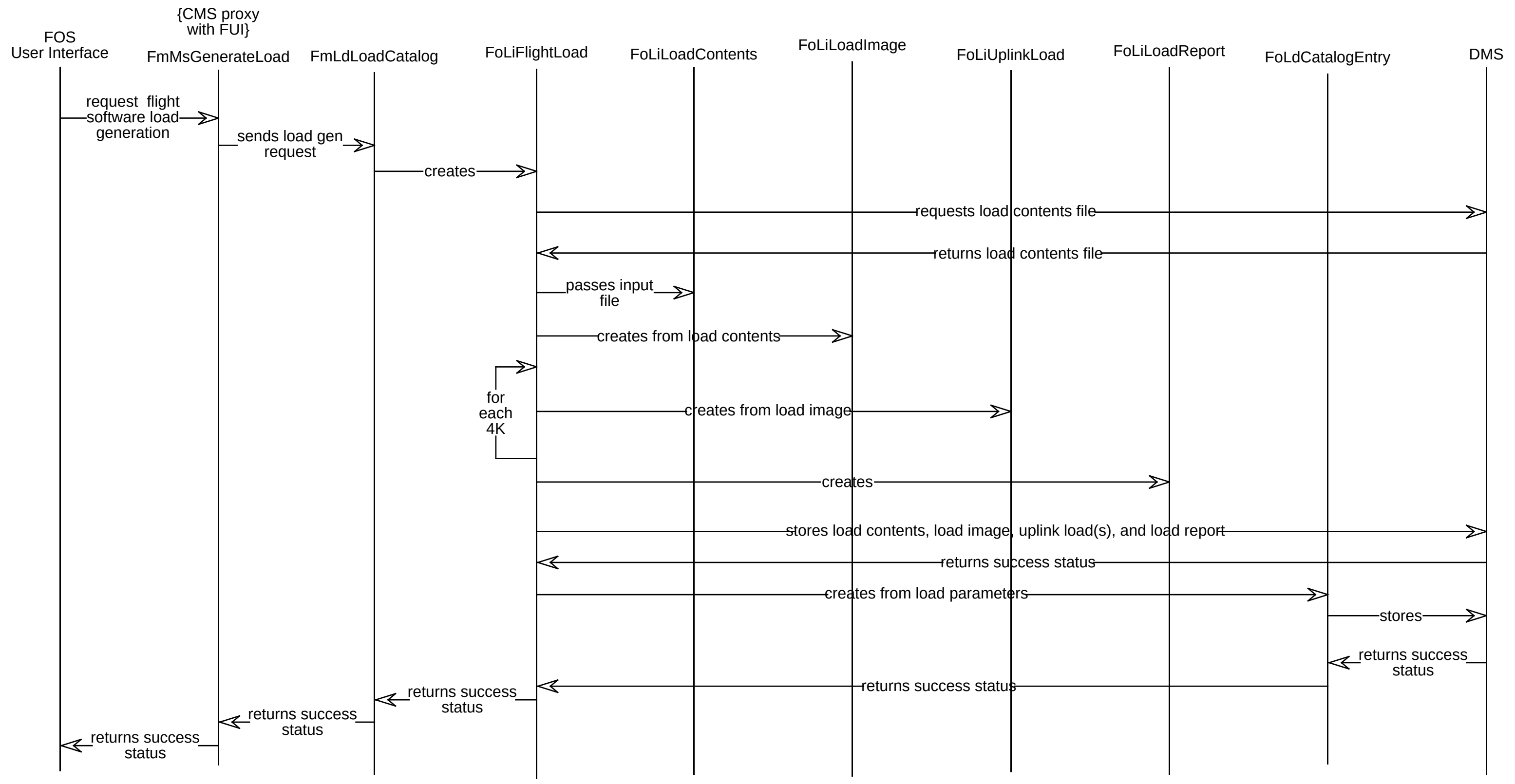


Figure 3.6-14. Flight Software Load Generation Event Trace

3.6.4.8 ATC Load Generation Scenario

3.6.4.8.1 ATC Load Generation Abstract

The ATC Load Generation scenario describes the generation of an ATC Load from a directive list. An uplinkable form of the ATC Load will be generated, as well as a Load Report, a Load Image, and a Load Contents file. The ATC Load will have with it a Load Catalog Entry with its pertinent information.

3.6.4.8.2 ATC Load Generation Summary Information

Interfaces:

- Schedule Controller
- DMS

Stimulus:

- Receipt of a directive list from the Schedule Controller

Desired Response:

- ATC Uplink Load, Load Image,Load Contents, and Load Report generated and stored with DMS
- ATC Load entered into ATC Load Catalog

Pre-Conditions:

- Load Catalog software has been initialized

Post-Conditions:

- Generated load is available for uplink

3.6.4.8.3 ATC Load Generation Description

Figure 3.6-15 shows the ATC Load Generation Event Trace. An FoLiATCLoad object is created and passed a list of commands. ATC Load converts these commands to their binary form and places them in a load contents file. ATC Load generates the load image from the load contents file. ATC Load generates the uplink load from the load image by packetizing the load image and converting the load image to the proper spacecraft format. ATC Load creates the load report. The report, uplink load, load image, and load contents are stored with DMS. ATC Schedule sends the ATC Load to Load Catalog via the proxy FmMsStoreATCLoad. Load Catalog creates a Load Catalog Entry and requests DMS to add it to the load catalog database. Load Catalog passes a success message back to ATC Schedule.

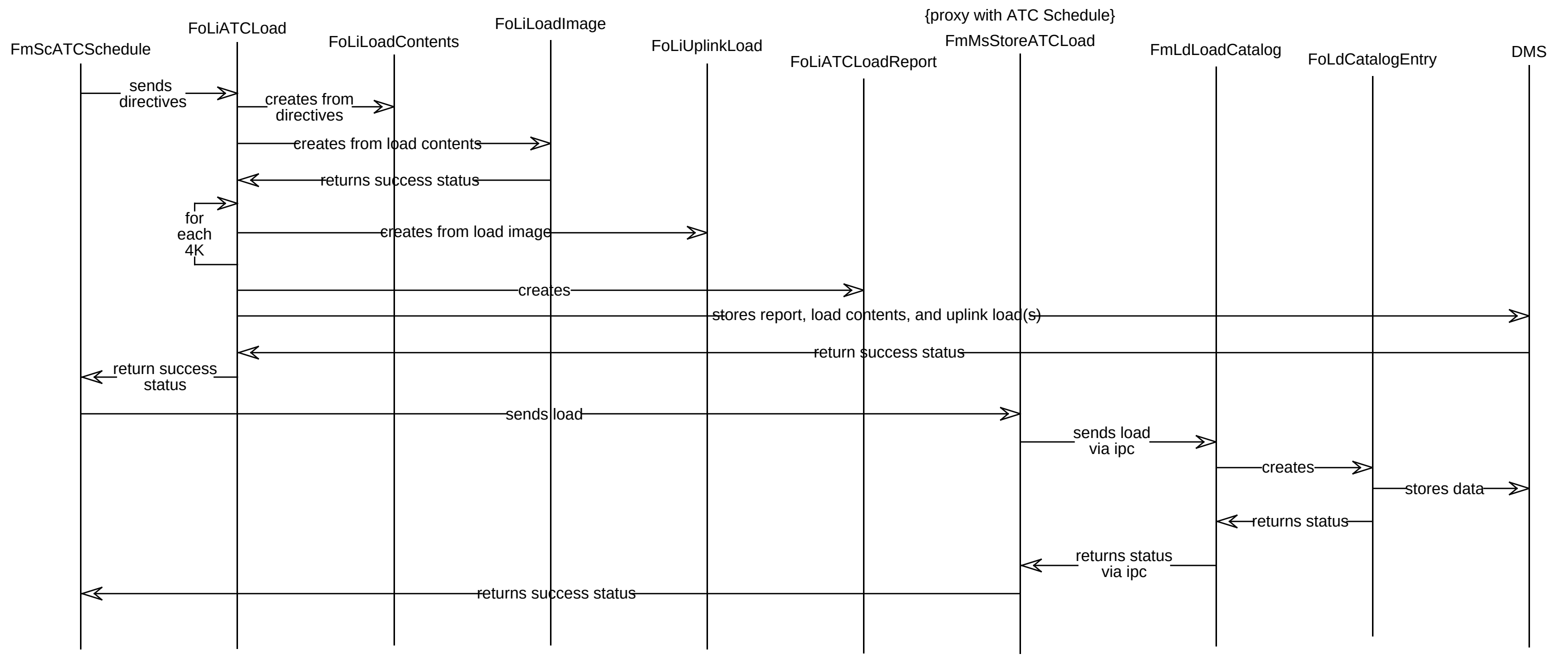


Figure 3.6-15. ATC Load Generation Event Trace

3.6.4.9 Uplink Notification Receipt Scenario

3.6.4.9.1 Uplink Notification Receipt Abstract

The Uplink Notification Receipt Scenario describes the updating of a Load Catalog Entry with the uplinked time of the load. Load Catalog receives an event message with the load name and time of uplink, and proceeds to update the appropriate Catalog Entry.

3.6.4.9.2 Uplink Notification Receipt Summary Information

Interfaces:

- DMS

Stimulus:

- Receipt of Load Uplink Status Event Message from DMS

Desired Response:

- Updated Load Catalog Entry stored with DMS

Pre-Conditions:

- Load Catalog software has been initiated

Post-Conditions:

- Update information sent to Spacecraft Model

3.6.4.9.3 Uplink Notification Receipt Description

Figure 3.6-16 shows the Uplink Notification Receipt Event Trace. Load Catalog receives a FoEvEvent message specifying the name of the load that has been uplinked and the time of uplink. Load Catalog creates a Catalog Entry object and requests DMS to retrieve the catalog entry for the specified load name. Load catalog updates the uplink time in the catalog entry's Uplink Info object. If this is the last uplink load (partition) for this load name, Load Catalog updates the uplink time in the Catalog Entry object and requests DMS to update the Catalog Entry for the load. Load Catalog creates a FmMsUpdateBuffer request and sends it to FmSmMapBuffer, the proxy to Spacecraft Model.

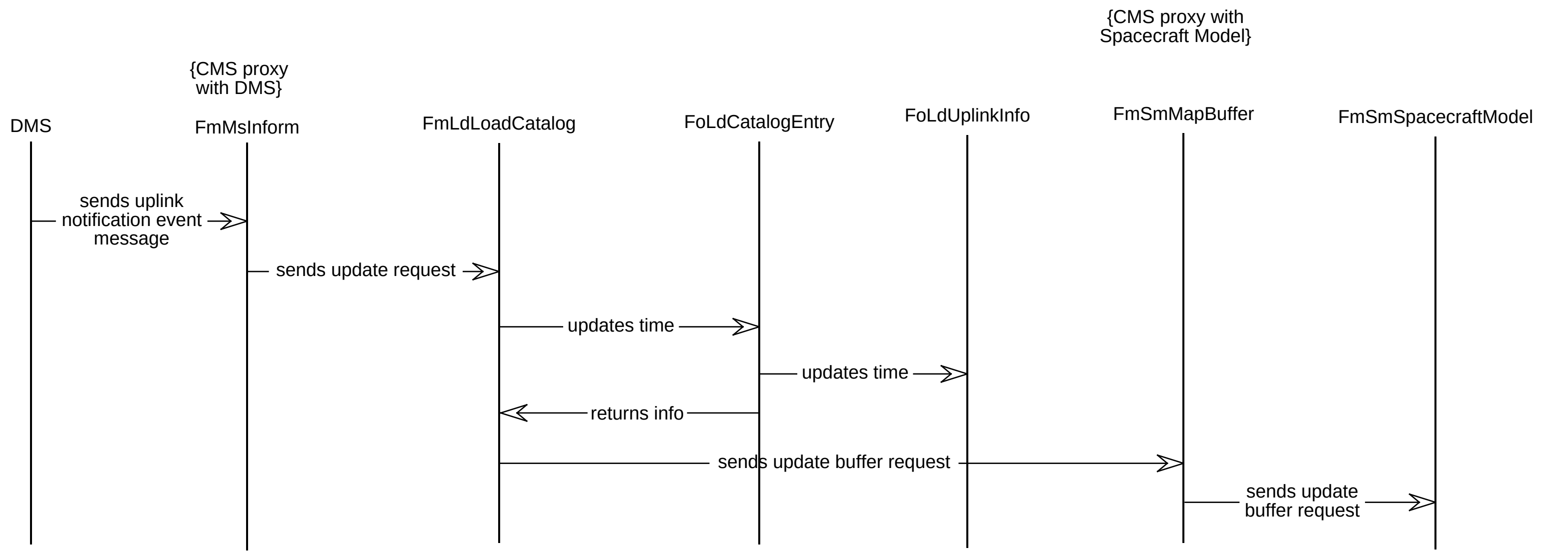


Figure 3.6-16. Uplink Notification Receipt Event Trace

3.6.5 Load Catalog Data Dictionary

System Include Files

`rw/collect.h`
`rw/dlistcol.h`

Include Files

`EcTypes.h`

Preprocessor Macros

`_FmLdLoadCatalog_h_`

Types

`class` **`FmLdLoadCatalog`**

class definition

Base Classes

`public` **`RWCollectable`**

Public Functions

`EcTInt` **`CheckForUplink`**`(EcTInt)`

Checks to see if a load with the given DAS id has been uplinked to the spacecraft. Returns TRUE if the load has been uplinked.

`EcTVoid` **`ConstraintCheck`**`(const FoMsRTSLoadGenReq&)`

Sends an RTS load contents file to the Command Model for constraint checking.

`EcTInt` **`CreateCatalogEntry`**`(const FoLiATCLoad&)`

After an ATC load has been generated, creates a load catalog entry from information in the load and stores both the entry and the load in the database.

`EcTInt` **`CreateCatalogEntry`**`(FoMsLoadGenReq*)`

After any load (except for ATC) has been successfully generated, creates a load catalog entry from information in the load gen request and stores the entry in the database.

`EcTVoid` **`HandleMessage`**`()`

Determines what type of message it has received, calls the appropriate process function, and sends the return value of that function to the proxy.

`EcTVoid` **`MakeFSWload`**`(const FoMsFSWLoadGenReq&)`

Constructs a flight software load from the request.

`EcTVoid` **`MakeMicroload`**`(const FoMsMPLoadGenReq&)`

Constructs a microprocessor load from the request.

`EcTVoid` **`MakeRTSload`**`(const FoMsRTSLoadGenReq&)`

Constructs a RTS load from the request.

`EcTVoid` **`MakeTableload`**`(const FoMsTableLoadGenReq&)`

Constructs a table load from the request.

`EcTInt` **`ProcessDeletions`**`(const RWSlistCollectables&)`

Deletes all loads on the input list from the load catalog.

`EcTVoid` **`ProcessLoadGenRequest`**`(const FoMsLoadGenReq&)`

Makes the appropriate load for the request, performs constraint checking as necessary, and returns a status.

`EcTVoid` **`ProcessLoadUplinkStatus`**`(const FoEvEvent&)`

Calls UpdateCatalogEntry() and passes in the event, and sends an update request to spacecraft model.

`EcTInt` **`ToFUI`**`(const FoMsCMSStatus&)`

Sends the input object to the proxy, which will send it to FUI.

`EcTInt` **`UpdateCatalogEntry`**`(const RWCString&)`

Fetches the appropriate catalog entry from the database, and increments the number of times scheduled.

`FmMsUpdateBuffer&` **`UpdateCatalogEntry`**`(const FoEvEvent&)`

Fetches the appropriate catalog entry from the database, and updates it with the information in the status.

Private Data

`FoEvEvent*` **`myEventPtr`**

A pointer to an event.

`FoMsCMSStatus` **`myProcessingStatus`**

The result of the processing of any load. This attribute will be changed after every load processing.

System Include Files

`rw/collect.h`

Preprocessor Macros

`_FmMsCatalogUpdate_h_`

Types

`class` **`FmMsCatalogUpdate`**

Base Classes

`public` **`RWCollectable`**

Public Functions

`EcTVoid` **`LoadScheduled`**`(const HString&)`

Called by PAS to notify Load Catalog that a certain load has been scheduled for uplink.

Private Functions

`EcTInt` **`CreateConnection`**`(void)`

Creates the two-way connection between this proxy and FmLdLoadCatalog.

`EcTVoid` **`DestroyConnection`**`(void)`

Destroys the connection between this proxy and FmLdLoadCatalog.

`EcTInt` **`Send`**`(const RWCString&)`

Sends an object to FmLdLoadCatalog via IPC.

System Include Files

`rw/collect.h`

Preprocessor Macros

`_FoMsGenInfo_h_`

Types

`class` **`FoMsGenInfo`**

class definition

Base Classes

`public` **`RWCollectable`**

Private Data

`RWCString` **`myDirectory`**

Directory where the table data is located.

`RWCString` **`myFilename`**

File that contains the table data.

`RWCString` **`myTableName`**

Name of the table for which the load is to be generated.

System Include Files

`rw/collect.h`

Preprocessor Macros

`_FmMsGenTable_h_`

Types

`class` **`FmMsGenTable`**

class definition

Base Classes

`public` **`RWCollectable`**

Public Functions

`EcTInt` **`MakeTableLoad`**`(const FoMsGenInfo&)`

Called by ANA. Sends the value of FoMsGenInfo to FmLdLoadCatalog via IPC requesting the generation of a table load and waits for a response of success or failure.

Private Functions

`EcTInt` **`CreateConnection`**`()`

Creates the two-way connection between this proxy and FmLdLoadCatalog.

`EcTVoid` **`DestroyConnection`**`()`

Destroys the connection between this proxy and FmLdLoadCatalog.

`FoMsCMSStatus&` **`Receive`**`()`

Receives an EcTInt from FmLdLoadCatalog via IPC and returns it.

`EcTVoid` **`Send`**`(const FoMsGenInfo&)`

Sends an object to FmLdLoadCatalog via IPC.

System Include Files

`rw/collect.h`

Preprocessor Macros

`_FmMsGenerateLoad_h_`

Types

`class` **`FmMsGenerateLoad`**

class definition

Base Classes

`public` **`RWCollectable`**

Public Functions

`FoMsCMSStatus&` **`ConstraintOverride`**`(option)`

Sends the value of option to FmLdLoadCatalog via IPC and waits for a response.

`EcTInt` **`CreateConnection`**`()`

Creates the two-way connection between this proxy and FmLdLoadCatalog.

`EcTVoid` **`DestroyConnection`**`()`

Destroys the connection between this proxy and FmLdLoadCatalog.

`FoMsCMSStatus&` **`GenerateLoad`**`(const FoMsLoadGenReq&)`

Called by FUI to send the load gen request. Returns a response.

Public Types

`enum` **`option`**

Enumerators

`n`
`y`

Private Functions

`FoMsCMSStatus&` **`Receive`**`()`

Receives an FoMsCMSStatus object from FmLdLoadCatalog via IPC and returns it.

`EcTVoid` **`Send`**`(const RWCollectable&)`

Sends an object to FmLdLoadCatalog via IPC.

Preprocessor Macros

`_FmMsInform_h_`

Types

class **FmMsInform**

Public Functions

`EcTVoid` **`InformCMS`**`(const FoEvEvent&)`

Called by DMS to send Load Catalog an event message. This operation is used to notify CMS that a load has been up-linked or that a load contents file has been imported into DMS.

Private Functions

`EcTInt` **`CreateConnection`**`(void)`

Creates the two-way connection between this proxy and FmLdLoadCatalog.

`EcTVoid` **`DestroyConnection`**`(void)`

Destroys the connection between this proxy and FmLdLoadCatalog.

`EcTVoid` **`Send`**`(const RWCollectable&)`

Sends an object to FmLdLoadCatalog via IPC.

Preprocessor Macros

`_FmMsStoreATCLoad_h_`

Types

class **FmMsStoreATCLoad**

class definition - This class represents an interface between the ATC Schedule and the Load Catalog. It uses ipc to relay information between this class and the Load Catalog. ATC Schedule sends information to this class via function calls.

Public Functions

`EcTInt` **`CheckForLoad`**`(EcTInt)`

Called by ATC Schedule to send a DAS Id to the Load Catalog and get back an integer status, indicating that the load associated with this DAS Id has or has not been uplinked.

`EcTInt` **`CreateConnection`**`()`

Creates the two-way connection between this proxy and FmLdLoadCatalog.

`EcTInt` **`DeleteLoads`**`(const RWSlistCollectables&)`

Called by ATC Schedule to delete all loads on the input list. Returns a response.

`EcTVoid` **`DestroyConnection`**`()`

Destroys the connection between this proxy and FmLdLoadCatalog.

`EcTInt` **`Receive`**`()`

Receives an FoMsCMSStatus object from FmLdLoadCatalog via IPC and returns it.

`EcTVoid` **`Send`**`(const RWCollectable&)`

Sends an object to FmLdLoadCatalog via IPC.

`EcTInt` **`StoreLoad`**`(const FoLiATCLoad&)`

Called by ATC Schedule to send a load for storage. Returns a response.

Preprocessor Macros

`_FmMsUpdateBuffer_h_`

Types

`class` **`FmMsUpdateBuffer`**

This class represents a message from the CMS load catalog to update the ATC/RTS buffers. It is sent from the load catalog when an uplink verification is received from the Command Subsystem It is also used to update the ground image.

Private Data

`EcTInt` **`myBufferID`**

Represent the buffer number that needs to be updated, pertains to RTS buffer number

`EcTInt` **`myEndLocation`**

Represent the End location in the buffer

`RWCString` **`myLoadName`**

Represents the load that was uplinked

`EcTInt` **`myStartLocation`**

Represents the start location in the buffer

`RWCString` **`myTableName`**

Represents the table name for the buffer update

`EcTInt` **`myType`**

represents the type of buffer affected by the uplink

Preprocessor Macros

`_FmMsValidateConstraints_h_`

Types

`class` **`FmMsValidateConstraints`**

This class represents the interface proxy class between CMS internal subsystems and the FmCcCommandModel class. FmCcCommandModel manages the command rule-based constraint checking.

Public Functions

`EcTInt` **`CreateConnection`**`(void)`

Establishes a connection with FmCcCommandModel to receive constraint checking request from the schedule controller and the load catalog

`EcTVoid` **`DestroyConnection`**`(void)`

Destroys the connection with FmCcCommandModel

`FoMsCMSStatus&` **`Receive`**`(void)`

Receives the results of rule-base command constraint checking, FoMsCMSStatus

`FoMsCMSStatus&` **`Send`**`(const RWCollectable&)`

Sends either a FmScConstCk command list from the schedule controller or a FoEcDirective list created from an RTS load contents file to the FmCcCommandModel for rule-base command constraint checking

`FoMsCMSStatus&` **`ValidateCommands`**`(const FmScConstCk&)`

FmScScheduleController invokes this function to send the DAS scheduled command list to be command rule-based constraint checked

`FoMsCMSStatus&` **`ValidateRTS`**`(const RWCString&, const RWCString&)`

FmLdLoadCatalog invokes this function to send the directory name and load name from the generate RTS load request to be command rule-based constraint checked. This function creates the FmMnDirectiveList to the FmCcCommandModel.

Preprocessor Macros

`_FmSmMapBuffer_h_`

Types

`class` **`FmSmMapBuffer`**

This class represents the interface proxy class between CMS internal subsystems and the FmSmSpacecraft class. FmSmSpacecraft manages the buffer modeling for ATC, RTS and table buffers and the ground

```
imaging.
```

Public Functions

`EcTInt` **`CreateConnection`**`(EcTVoid)`

Establishes a connection with FmSmSpacecraft to receive requests from the schedule controller and the load catalog

`EcTVoid` **`DeleteBuffers`**`(const RWSlistCollectables&)`

Request received from load catalog when a late change as been successfully processed. The predicted buffer models associated with all of the generated loads are deleted. Instantiates an FmMsDeleteATCBuffers object.

`EcTVoid` **`Destroy`**`(EcTVoid)`

Destroys the connection with FmCcCommandModel

`FmMsATCBufferInfo` **`GetATCBufStartTime`**`(const FoEcTime&)`

Requests the start time of the 1st command in the buffer that will be used to model the newly received DAS or late change request

`RWSlistCollectables&` **`MapATC`**`(const FmMnDirectiveList&, const FOSTimeInterval&, const FoEcTime&, const EcTInt&)`

Request FmSmSpacecraft to map the command list into an ATC buffer model. Instantiates an FmMsATCMapRequest object to be sent to FmSmSpacecraft.

`RWSlistCollectables&` **`MapLateChange`**`(const FmMnDirectiveList&, const FOSTimeInterval&, const FoEcTime&, const EcTInt&)`

Requests FmSmSpacecraft to map the late change command list into the correct buffer model. Instantiates an FmMsATCMapRequest object to be sent to FmSmSpacecraft.

`RWSlistCollectables&` **`Receive`**`(EcTVoid)`

Receives the response from FmSmSpacecraftModel It receives either A list of FmMsLoadData objects or a FmMsATCBufferInfo object

`EcTVoid` **`Send`**`(const RWCollectable&)`

Sends messages to FmSmSpacecraftModel. Sends FmMsATCMapRequest, FmMsDeleteATCBuffers, or FmMsUpdateBuffer.

`EcTVoid` **`UpdateBuffer`**`(const FmMsUpdateBuffer&)`

Request the buffer be updated to a new status

System Include Files

`rw/collect.h`

Preprocessor Macros

`_FoFmDataField_h_`

Types

`class` **`FoFmDataField`**

class definition

Base Classes

`public` **`RWCollectable`**

Public Functions

`RWBitVec` **`ProduceBinary`**`()`

Produces the binary form of this field.

Private Data

`RWCString` **`myDataUnits`**

The units of the field value.

`RWCString` **`myFieldDescriptor`**

Textual information describing the field and its value.

`EcTInt` **`myFieldNumber`**

A unique value which identifies the field within the table.

`EcTInt` **`myRangeCheckFlag`**

An indicator of whether range checking is to be performed.

`EcTInt` **`myScaleFactor`**

The scale factor to be applied to the word value within this field.

`EcTInt` **`myTableNumber`**

A unique value specifying a memory table.

`EcTInt` **`myValueBitSize`**

The size of the value in bits.

`EcTInt` **`myValueOverrideFlag`**

An indicator of whether the value may be overwritten with a new value during table generation.

`RWCString` **`myValueType`**

The data type of the value in this field.

Include Files

`FdDbAccessor.h`

Preprocessor Macros

`_FoFmTableFormat_h_`

Types

`class` **`FoFmTableFormat`**

class definition

Base Classes

`public` **`FdDbAccessor`**

Public Functions

`RWBitVec` **`ProduceBinary`**`()`

Produces the binary form of each field in the table.

Private Data

`RWCString` **`myDescriptor`**

Textual information describing the table.

`EcTInt` **`myMaxSize`**

The maximum number of words allowed in the table.

`EcTInt` **`myStartAddress`**

The starting location in memory for the table.

`RWCString` **`myTableMnemonic`**

The name that is used to reference the table.

`EcTInt` **`myTableNumber`**

A unique value identifying the table.

`RWCString` **`myTableType`**

The type of memory table.

Include Files

`FdDbAccessor.h`

Preprocessor Macros

`_FoLdCatalogEntry_h_`

Types

`class` **`FoLdCatalogEntry`**

class definition

Base Classes

public **FdDbAccessor**

Private Data

EcTInt **myDASId**

The DASId which the load covers. This is only valid for ATC loads.

EcTInt **myEndLocation**

The last location in memory used by the load.

RWCString **myLoadName**

The name of the load.

EcTInt **myLoadSize**

The number of bytes in the load.

RWCString **myLoadType**

The load type. ATC, RTS, TAB, MP, or FSW.

EcTInt **myNumTimesSchd**

The number of times that the load has been scheduled for uplink.

RWCString **myOwner**

The owner of the load.

EcTInt **myRTSBuffer**

The number of the buffer in which the RTS load is going to reside.

RWCString **mySpacecraftLocation**

The location of the load on the spacecraft.

EcTInt **myStartLocation**

The first location in memory used by the load.

RWCString **myStorageLocation**

The location of the load in storage on the ground.

RWSlistCollectables **myUplinkLoads**

A list of uplink loads associated with this load.

RWTime **myUplinkTime**

The time at which the load was actually uplinked to the spacecraft.

FOSTimeInterval **myValidUplinkPeriod**

The valid uplink period of the load.

Include Files

FoLiLoadReport.h

Preprocessor Macros

_FoLiATCLoadReport_h_

Types

class **FoLiATCLoadReport**

class definition

Base Classes

public **FoLiLoadReport**

Private Data

FmMnDirectiveList **myCommandList**

The entire list of commands which constitute the load.

FmMnDirectiveList **myControlCommands**

The control commands contained in the load.

RWTime **myStartTime**

The time of the first command in the load.

RWTime **myStopTime**

the time of the last command in the load.

Include Files

FoLiLoad.h

Preprocessor Macros

_FoLiFlightLoad_h_

Types

class **FoLiFlightLoad**

stp/omt class definition 174512 - This class represents a flight software load. The class maintains load-related information. The class contains behaviors necessary to produce the load's uplink form and load report.

Base Classes

public **FoLiLoad**

Public Functions

EcTInt **BuildUplinkLoad**()

Formats the load contents into an appropriate storage command and generates the load report.

Private Data

EcTInt **myEndingLocation**

The last memory location used by the load.

EcTInt **myMemoryUpdateSize**

The size of the load in bytes.

EcTInt **myStartingLocation**

The first location in memory used by the load.

System Include Files

`rw/collect.h`

Preprocessor Macros

`_FoLiLoad_h_`

Types

`class` **`FoLiLoad`**

Base Classes

`public` **`RWCollectable`**

Public Functions

`virtual EcTInt` **`BuildUplinkLoad`**`(const FoLiLoadImage&)`

Builds the uplinkable load and the load report.

`FoMsCMSStatus&` **`CreateLoad`**`(const FoMsLoadGenReq&)`

Reads in the file from the request and creates the load.

`EcTInt` **`GenerateLoadImage`**`(const FoLiLoadContents&)`

Generates the binary for the load and stores it in a file.

Private Data

`RWCString` **`myDestination`**

The destination on the spacecraft for the load.

`RWCString` **`myDirectory`**

The directory where the load contents file from which the load is generated exists.

`FoLiLoadContents` **`myLoadContents`**

The load contents object.

`RWCString` **`myLoadName`**

The name of the load.

`EcTInt` **`myLoadSize`**

The size of the load in bytes.

`EcTInt` **`myNumberOfPieces`**

The number of uplink loads for this load.

`RWCString` **`myOwner`**

The id of the owner of the load.

`EcTInt` **`mySizeOfLastPiece`**

The number of bytes of the last uplinkable load.

`EcTInt` **`mySpacecraftId`**

The id of the spacecraft for which the load is valid.

`FoMsCMSStatus` **`myStatus`**

The processing status of the load.

`RWSlistCollectables` **`myUplinkLoads`**

The uplinkable portions of the load.

FOSTimeInterval **myUplinkPeriod**

The uplink period of the load.

Include Files

FoDsFile.h

Preprocessor Macros

_FoLiLoadContents_h_

Types

class **FoLiLoadContents**

class definition - This class represents a file into which the load contents are placed. The load contents are sent directly to CMS from an external interface, except in the case of ATC.

Base Classes

public **FoDsFile**

Include Files

FoDsFile

Preprocessor Macros

_FoLiLoadReport_h_

Types

class **FoLiLoadReport**

class definition

Base Classes

public **FoDsFile**

Private Data

EcTInt **myEndLocation**

The last memory location used by the load.

RWCString **myLoadName**

The name of the load for which this report was written.

EcTInt **mySize**

The size of the load in bytes.

EcTInt **myStartLocation**

The first memory location used by the load.

RWCString **myType**

the type of the load for which this report was written.

FOSTimeInterval **myUplinkPeriod**

The uplink window of the load for which this report was written.

Include Files

`FoLiLoad.h`

Preprocessor Macros

`_FoLiMicroLoad_h_`

Types

`class` **`FoLiMicroLoad`**

stp/omt class definition 174512 - This class represents a microprocessor load. The class maintains load-related information. The class contains behavior necessary to produce the load's uplink form and load report.

Base Classes

`public` **`FoLiLoad`**

Public Functions

`EcTInt` **`BuildUplinkLoad`**`()`

Formats the load contents into an appropriate storage command and generates the load report.

Private Data

`EcTInt` **`myEndingLocation`**

The last memory location used by the load.

`EcTInt` **`myMemoryUpdateSize`**

The size of the load in bytes.

`EcTInt` **`myStartingLocation`**

The first location in memory used by the load.

Include Files

`FoLiLoad.h`

Preprocessor Macros

`_FoLiRTSLoad_h_`

Types

`class` **`FoLiRTSLoad`**

class definition

Base Classes

`public` **`FoLiLoad`**

Public Functions

`EcTInt` **`BuildUplinkLoad`**`(const FoMsLoadGenReq&)`

Builds the uplinkable load and the load report.

`FoMsCMSStatus&` **`CreateLoad`**`(const FoMsLoadGenReq&)`

Creates the load from the load gen request and returns a status.

`EcTInt` **`GenerateBinary`**`(RWFile&)`

Creates the binary for each command in the command list.

Private Data

`RWDlistCollectables` **`myCriticalCommands`**

A list of commands in the load which are flagged as critical.

`RWDlistCollectables` **`myDirectiveList`**

The entire list of directives which define the load.

`EcTInt` **`myRTSBuffDestination`**

The buffer number in which the load will reside on the spacecraft.

Include Files

`FoLiLoad.h`

Preprocessor Macros

`_FoLiTableLoad_h_`

Types

`class` **`FoLiTableLoad`**

class definition

Base Classes

`public` **`FoLiLoad`**

Public Functions

`EcTInt` **`BuildUplinkLoad`**`()`

Constructs the uplinkable form of the load.

`EcTInt` **`ComposeReport`**`()`

Populates the load report with the pertinent information about the load.

`EcTInt` **`CreateLoad`**`(const FoMsTableLoadGenReq&)`

Generates the uplinkable load based on the request.

`EcTInt` **`GenerateLoadImage`**`(const FoLiLoadContents&)`

Generates the load image from the contents.

`FoFmTableFormat` **`RetrieveTableFormat`**`(const RWCString&)`

Retrieves the appropriate format for this table from DMS.

Private Data

`EcTInt` **`myEndLocation`**

The ending location in memory for the table load.

`FoLiLoadReport` **`myLoadReport`**

The load report associated with this load.

`EcTInt` **`myStartLocation`**

The starting location in memory for the table load.

`RWCString` **`myTableName`**

The name of the table for which this load is valid. Table name is used to find the format in DMS.

Include Files

`FoDsFile.h`

Preprocessor Macros

`_FoLiUplinkLoad_h_`

Types

`class` **`FoLiUplinkLoad`**

class definition

Base Classes

`public` **`FoDsFile`**

Public Functions

`EcTInt` **`BuildLoad`**`(const FoLiLoadImage&)`

Generates the CRC for the load, puts the load into packets, and stores the load with DMS.

`EcTInt*` **`BuildLoadData`**`(EcTInt*)`

Sets the command destination information and fills in the command data.

`EcTInt*` **`CCSDSWrap`**`(EcTInt*)`

Generates the packets for the load.

System Include Files

`rw/collect.h`

Preprocessor Macros

`_FoMsCMSStatus_h_`

Types

`class` **`FoMsCMSStatus`**

class definition

Base Classes

`public` **`RWCollectable`**

Private Data

`EcTInt` **`myId`**

The id of this message.

`RWCString` **`myStatus`**

Pertinent information about the status object. Mostly used to explain why a process failed.

Include Files

`FoMsLoadGenReq.h`

Preprocessor Macros

`_FoMsFSWLoadGenReq_h_`

Types

`class` **`FoMsFSWLoadGenReq`**

stp/omt class definition 2795027

Base Classes

`public` **`FoMsLoadGenReq`**

System Include Files

`rw/collect.h`

Preprocessor Macros

`_FoMsLoadGenReq_h_`

Types

`class` **`FoMsLoadGenReq`**

Base Classes

`public` **`RWCollectable`**

Private Data

`RWCString` **`myDirectory`**

The directory in which the input file is stored.

`RWCString` **`myFunction`**

The purpose of the load.

`RWCString` **`myLoadName`**

The name which should be assigned to the load. Also the name of the input file.

`EcTInt` **`mySize`**

The size of the input file in bytes.

`RWCString` **`mySpacecraftId`**

The id of the spacecraft for which the load should be made.

`RWCString` **`mySubsystemId`**

The id of the subsystem for which the load should be made.

`RWCString` **`myUserId`**

The identifying information about the originator of the request.

`FOSTimeInterval` **`myValidUplinkPeriod`**

The period during which the load is valid.

Include Files

`FoMsLoadGenReq.h`

Preprocessor Macros

`_FoMsMPLoadGenReq_h_`

Types

`class` **`FoMsMPLoadGenReq`**

stp/omt class definition 2795028

Base Classes

`public` **`FoMsLoadGenReq`**

Include Files

`FoMsLoadGenReq.h`

Preprocessor Macros

`_FoMsRTSLoadGenReq_h_`

Types

`class` **`FoMsRTSLoadGenReq`**

class definition

Base Classes

`public` **`FoMsLoadGenReq`**

Private Data

`EcTInt` **`myCheckOnlyFlag`**

Specifies if the RTS is to be constraint checked only.

`EcTInt` **`myOffset`**

The offset of the commands in the RTS.

`EcTInt` **`myRTSBufferNumber`**

The buffer number in which the RTS load will be stored.

Include Files

`FoMsLoadGenReq.h`

Preprocessor Macros

`_FoMsTableLoadGenReq_h_`

Types

`class` **`FoMsTableLoadGenReq`**

class definition

Base Classes

public **FoMsLoadGenReq**

Private Data

EcTInt **myEndLocation**

The location in the table where the load should end. If it is not set, the table load will include all locations from myStart-Field to the end of the table.

EcTInt **myStartField**

The location in the table where the load should begin. This makes it possible to have partial table loads. The default is set to 0.

RWCString **myTableName**

The name of the table, used to locate the format.

Preprocessor Macros

_FpRmLoadActDel_h_

Types

class **FpRmLoadActDel**

Instances of a FpRmLoadActDel provide the proxy for sending load deletion messages.

Public Construction

FpRmLoadActDel(const FpRmLoadActDel&)

Copy constructor.

FpRmLoadActDel(void)

Constructor.

~FpRmLoadActDel(void)

Destructor.

Public Functions

int **loadActDelRequest**(const HString&, int)

Send the request saying that a load has been deleted.

void **operator =**(const FpRmLoadActDel&)

Assignment operator.

Abbreviations and Acronyms

ACL	Access Control List
AD	Acceptance Check/TC Data
AGS	ASTER Ground System
AM	Morning (ante meridian) -- see EOS AM
Ao	Availability
APID	Application Identifier
ARAM	Automated Reliability/Availability/Maintainability
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer (formerly ITIR)
ATC	Absolute Time Command
BAP	Baseline Activity Profile
BC	Bypass check/Control Commands
BD	Bypass check/TC Data (Expedited Service)
BDU	Bus Data Unit
bps	bits per second
CAC	Command Activity Controller
CCB	Change Control Board
CCSDS	Consultative Committee for Space Data Systems
CCTI	Control Center Technology Interchange
CD-ROM	Compact Disk-Read Only Memory
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CERES	Clouds and Earth's Radiant Energy System
CI	Configuration item
CIL	Critical Items List
CLCW	Command Link Control Words
CLTU	Command Link Transmission Unit
CMD	Command subsystem
CMS	Command Management Subsystem
CODA	Customer Operations Data Accounting
COP	Command Operations Procedure
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit

CRC	Cyclic Redundancy Code
CSCI	Computer software configuration item
CSMS	Communications and Systems Management Segment
CSS	Communications Subsystem (CSMS)
CSTOL	Customer System Test and Operations Language
CTIU	Command and Telemetry Interface Unit (AM-1)
DAAC	Distributed Active Archive Center
DAR	Data Acquisition Request
DAS	Detailed Activity Schedule
DAT	Digital Audio Tape
DB	Data Base
DBA	Database Administrator
DBMS	Database Management System
DCE	Distributed Computing Environment
DCP	Default Configuration Procedure
DEC	Digital Equipment Corporation
DES	Data Encryption Standard
DFCD	Data Format Control Document
DID	Data Item Description
DMS	Data Management Subsystem
DOD	Digital Optical Data
DoD	Department of Defense
DS	Data Server
DSN	Deep Space Network
DSS	Decision Support System
e-mail	electronic mail
Ecom	EOS Communication
ECS	EOSDIS Core System
EDOS	EOS Data and Operations System
EDU	EDOS Data Unit
EGS	EOS Ground System
EOC	Earth Observation Center (Japan); EOS Operations Center (ECS)
EOD	Entering Orbital Day
EON	Entering Orbital Night
EOS	Earth Observing System

EOSDIS	EOS Data and Information System
EPS	Encapsulated Postscript
ESH	EDOS Service Header
ESN	EOSDIS Science Network
ETS	EOS Test System
EU	Engineering Unit
EUVE	Extreme Ultra Violet Explorer
FAS	FOS Analysis Subsystem
FAST	Fast Auroral Snapshot Explorer
FDDI	Fiber Distributed Data Interface
FDF	Flight Dynamics Facility
FDIR	Fault Detection and Isolation Recovery
FDM	FOS Data Management Subsystem
FMEA	Failure Modes and Effects Analyses
FOP	Frame Operations Procedure
FORMATS	FDF Orbital and Mission Aids Transformation System
FOS	Flight Operations Segment
FOT	Flight Operations Team
FOV	Field-Of-View
FPS	Fast Packet Switch
FRM	FOS Resource Management Subsystem
FSE	FOT S/C Evolutions
FTL	FOS Telemetry Subsystem
FUI	FOS User Interface
GB	Gigabytes
GCM	Global Circulation Model
GCMR	Global Circulation Model Request
GIMTACS	GOES I-M Telemetry and Command System
GMT	Greenwich Mean Time
GN	Ground Network
GOES	Geostationary Operational Environmental Satellite
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
H&S	Health and Safety
H/K	Housekeeking
HST	Hubble Space Telescope

I/F	Interface
I/O	Input/Output
ICC	Instrument Control Center
ICD	Interface Control Document
ID	Identifier
IDB	Instrument Database
IDR	Incremental Design Review
IEEE	Institute of Electrical and Electronics Engineers
IOT	Instrument Operations Team
IP	International Partners
IP-ICC	International Partners-Instrument Control Center
IPs	International Partners
IRD	Interface requirements document
ISDN	Integrated Systems Digital Network
ISOLAN	Isolated Local Area Network
ISR	Input Schedule Request
IST	Instrument Support Terminal
IST	Instrument Support Toolkit
IWG	Investigator Working Group
JPL	Jet Propulsion Laboratory
Kbps	Kilobits per second
LAN	Local Area Network
LaRC	Langley Research Center
LASP	Laboratory for Atmospheric Studies Project
LEO	Low Earth Orbit
LOS	Loss of Signal
LSM	Local System Manager
LTIP	Long-Term Instrument Plan
LTSP	Long-Term Science Plan
MAC	Medium Access Control; Message Authentication Code
MB	Megabytes
MBONE	Multicast Backbone
Mbps	Megabits per second
MDT	Mean Down Time
MIB	Management Information Base

MISR	Multi-angle Imaging Spectro-Radiometer
MMM	Minimum, Maximum, and Mean
MO&DSD	Mission Operations and Data Systems Directorate (GSFC Code 500)
MODIS	Moderate resolution Imaging Spectrometer
MOPITT	Measurements Of Pollution In The Troposphere
MSS	Management Subsystem
MTPE	Mission to Planet Earth
NASA	National Aeronautics and Space Administration
Nascom	NASA Communications Network
NASDA	National Space Development Agency (Japan)
NCAR	National Center for Atmospheric Research
NCC	Network Control Center
NEC	North Equator Crossing
NFS	Network File System
NOAA	National Oceanic and Atmospheric Administration
NSI	NASA Science Internet
NTT	Nippon Telephone and Telegraph
OASIS	Operations and Science Instrument Support
ODB	Operational Database
ODM	Operational Data Message
OMT	Object Model Technique
OO	Object Oriented
OOD	Object Oriented Design
OpLAN	Operational LAN
OSI	Open System Interconnect
PACS	Polar Acquisition and Command System
PAS	Planning and Scheduling
PDB	Project Data Base
PDF	Publisher's Display Format
PDL	Program Design Language
PDR	Preliminary Design Review
PI	Principal Investigator
PI/TL	Principal Investigator/Team Leader
PID	Parameter ID
PIN	Password Identification Number
POLAR	Polar Plasma Laboratory

POP	Polar-Orbiting Platform
POSIX	Portable Operating System for Computing Environments
PSAT	Predicted Site Acquisition Table
PSTOL	PORTS System Test and Operation Language
Q/L	Quick Look
R/T	Real-Time
RAID	Redundant Array of Inexpensive Disks
RCM	Real-Time Contact Management
RDBMS	Relational Database Management System
RMA	Reliability, Maintainability, Availability
RMON	Remote Monitoring
RMS	Resource Management Subsystem
RPC	Remote Processing Computer
RTCS	Relative Time Command Sequence
RTS	Relative Time Sequence; Real-Time Server
S/C	Spacecraft
SAR	Schedule Add Requests
SCC	Spacecraft Controls Computer
SCF	Science Computing Facility
SCL	Spacecraft Command Language
SDF	Software Development Facility
SDPS	Science Data Processing Segment
SDVF	Software Development and Validation Facility
SEAS	Systems, Engineering, and Analysis Support
SEC	South Equator Crossing
SLAN	Support LAN
SMA	S-band Multiple Access
SMC	Service Management Center
SN	Space Network
SNMP	System Network Mgt Protocol
SQL	Structured Query Language
SSA	S-band Single Access
SSIM	Spacecraft Simulator
SSR	Solid State Recorder
STOL	System Test and Operations Language

T&C	Telemetry and Command
TAE	Transportable Applications Environment
TBD	To Be Determined
TBR	To Be Replaced/Resolved/Reviewed
TCP	Transmission Control Protocol
TD	Target Day
TDM	Time Division Multiplex
TDRS	Tracking and Data Relay Satellite
TDRSS	Tracking and Data Relay Satellite System
TIROS	Television Infrared Operational Satellite
TL	Team Leader
TLM	Telemetry subsystem
TMON	Telemetry Monitor
TOO	Target Of Opportunity
TOPEX	Topography Ocean Experiment
TPOCC	Transportable Payload Operations Control Center
TRMM	Tropical Rainfall Measuring Mission
TRUST	TDRSS Resource User Support Terminal
TSS	TDRSS Service Session
TSTOL	TRMM System Test and Operations Language
TW	Target Week
U.S.	United States
UAV	User Antenna View
UI	User Interface
UPS	User Planning System
US	User Station
UTC	Universal Time Code; Universal Time Coordinated
VAX	Virtual Extended Address
VMS	Virtual Memory System
W/S	Workstation
WAN	Wide Area Network
WOTS	Wallops Orbital Tracking Station
XTE	X-Ray Timing Explorer

This page intentionally left blank.

Glossary

GLOSSARY of TERMS for the Flight Operations Segment

activity	A specified amount of scheduled work that has a defined start date, takes a specific amount of time to complete, and comprises definable tasks.
analysis	Technical or mathematical evaluation based on calculation, interpolation, or other analytical methods. Analysis involves the processing of accumulated data obtained from other verification methods.
attitude data	Data that represent spacecraft orientation and onboard pointing information. Attitude data includes: o Attitude sensor data used to determine the pointing of the spacecraft axes, calibration and alignment data, Euler angles or quaternions, rates and biases, and associated parameters. o Attitude generated onboard in quaternion or Euler angle form. o Refined and routine production data related to the accuracy or knowledge of the attitude.
availability	A measure of the degree to which an item is in an operable and committable state at the start of a "mission" (a requirement to perform its function) when the "mission" is called for an unknown (random) time. (Mathematically, operational availability is defined as the mean time between failures divided by the sum of the mean time between failures and the mean down time [before restoration of function].

availability (inherent) (A_i)	The probability that, when under stated conditions in an ideal support environment without consideration for preventive action, a system will operate satisfactorily at any time. The "ideal support environment" referred to, exists when the stipulated tools, parts, skilled work force manuals, support equipment and other support items required are available. Inherent availability excludes whatever ready time, preventive maintenance downtime, supply downtime and administrative downtime may require. A_i can be expressed by the following formula: A_i = MTBF/ (MTBF + MTTR) Where: MTBF = Mean Time Between Failures MTTR = Mean Time To Repair
availability (operational) (A_o)	The probability that a system or equipment, when used under stated conditions in an actual operational environment, will operate satisfactorily when called upon. A_o can be expressed by the following formula: A_o = MTBM / (MTBM + MDT + ST) Where: MTBM = Mean Time Between Maintenance (either corrective or preventive) MDT = Mean Maintenance Down Time where corrective, preventive administrative and logistics actions are all considered. ST = Standby Time (or switch over time)
	A schedule of activities for a target week corresponding to normal instrument operations constructed by integrating long term plans (i.e., LTSP, LTIP, and long term spacecraft operations plan).
build	An assemblage of threads to produce a gradual buildup of system capabilities.
calibration	The collection of data required to perform calibration of the instrument science data, instrument engineering data, and the spacecraft engineering data. It includes pre-flight calibration measurements, in-flight calibrator measurements, calibration equation coefficients derived from calibration software routines, and ground truth data that are to be used in the data calibration processing routine.

command	Instruction for action to be carried out by a space-based instrument or spacecraft.
command and data handling (C&DH)	The spacecraft command and data handling subsystem which conveys commands to the spacecraft and research instruments, collects and formats spacecraft and instrument data, generates time and frequency references for subsystems and instruments, and collects and distributes ancillary data.
command group	A logical set of one or more commands which are not stored onboard the spacecraft and instruments for delayed execution, but are executed immediately upon reaching their destination on board. For the U.S. spacecraft, from the perspective of the EOS Operations Center (EOC), a preplanned command group is preprocessed by, and stored at, the EOC in preparation for later uplink. A real-time command group is unplanned in the sense that it is not preprocessed and stored by the EOC.
detailed activity schedules	The schedule for a spacecraft and instruments which covers up to a10 day period and is generated/updated daily based on the instrument activity listing for each of the instruments on the respective spacecraft. For a spacecraft and instrument schedule the spacecraft subsystem activity specifications needed for routine spacecraft maintenance and/or for supporting instruments activities are incorporated in the detailed activity schedule.
direct broadcast	Continuous down-link transmission of selected real-time data over a broad area (non-specific users).

EOS Data and Operations System (EDOS) production data set	Data sets generated by EDOS using raw instrument or spacecraft packets with space-to-ground transmission artifacts removed, in time order, with duplicate data removed, and with quality/ accounting (Q/A) metadata appended. Time span or number of packets encompassed in a single data set are specified by the recipient of the data. These data sets are equivalent to Level 0 data formatted with Q/A metadata. For EOS, the data sets are composed of: instrument science packets, instrument engineering packets, spacecraft housekeeping packets, or onboard ancillary packets with quality and accounting information from each individual packet and the data set itself and with essential formatting information for unambiguous identification and subsequent processing.
housekeeping data	The subset of engineering data required for mission and science operations. These include health and safety, ephemeris, and other required environmental parameters.
instrument	o A hardware system that collects scientific or operational data. o Hardware-integrated collection of one or more sensors contributing data of one type to an investigation. o An integrated collection of hardware containing one or more sensors and associated controls designed to produce data on/in an observational environment.
instrument activity deviation list	An instrument's activity deviations from an existinginstrument activity list, used by the EOC for developing the detailed activity schedule.
instrument activity list	An instrument's list of activities that nominally covers seven days, used by the EOC for developing the detailed activity schedule.
instrument engineering data	Subset of telemetered engineering data required for performing instrument operations and science processing
instrument microprocessor memory loads	Storage of data into the contents of the memory of an instrument’s microprocessor, if applicable. These loads could include microprocessor-stored tables, microprocessor-stored commands, or updates to microprocessor software.

instrument resource deviation list	An instrument's anticipated resource deviations from anexisting resource profile, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
instrument resource profile	Anticipated resource needs for an instrument over a targetweek, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
instrument science data	Data produced by the science sensor(s) of an instrument, usually constituting the mission of that instrument.
long-term instrument plan (LTIP)	The plan generated by the instrument representative to the spacecraft's IWG with instrument-specific information to complement the LTSP. It is generated or updated approximately every six months and covers a period of up to approximately 5 years.
long-term science plan (LTSP)	The plan generated by the spacecraft's IWG containing guidelines, policy, and priorities for its spacecraft and instruments. The LTSP is generated or updated approximately every six months and covers a period of up to approximately five years.
long term spacecraft operations plan	Outlines anticipated spacecraft subsystem operations and maintenance, along with forecasted orbit maneuvers from the Flight Dynamics Facility, spanning a period of several months.
mean time between failure (MTBF)	The reliability result of the reciprocal of a failure rate that predicts the average number of hours that an item, assembly or piece part will operate within specific design parameters. (MTBF=1/(l) failure rate; (l) failure rate = # of failures/ operating time.
mean down time (MDT)	Sum of the mean time to repair MTTR plus the average logistic delay times.
mean time between maintenance (MTBM)	The mean time between preventive maintenance (MTBPM) and mean time between corrective maintenance (MTBCM) of the ECS equipment. Each will contribute to the calculation of the MTBM and follow the relationship: 1/MTBM = 1/MTBPM + 1/MTBCM
mean time to repair (MTTR)	The mean time required to perform corrective maintenance to restore a system/equipment to operate within design parameters.

object	Identifiable encapsulated entities providing one or more services that clients can request. Objects are created and destroyed as a result of object requests. Objects are identified by client via unique reference.
orbit data	Data that represent spacecraft locations. Orbit (or ephemeris) data include: Geodetic latitude, longitude and height above an adopted reference ellipsoid (or distance from the center of mass of the Earth); a corresponding statement about the accuracy of the position and the corresponding time of the position (including the time system); some accuracy requirements may be hundreds of meters while other may be a few centimeters.
playback data	Data that have been stored on-board the spacecraft for delayed transmission to the ground.
preliminary resource schedule	An initial integrated spacecraft schedule, derived from instrument and subsystem resource needs, that includes the network control center TDRSS contact times and nominally spans seven days.
preplanned stored command	A command issued to an instrument or subsystem to be executed at some later time. These commands will be collected and forwarded during an available uplink prior to execution.
principal investigator (PI)	An individual who is contracted to conduct a specific scientific investigation. (An instrument PI is the person designated by the EOS Program as ultimately responsible for the delivery and performance of standard products derived from an EOS instrument investigation.)
prototype	Prototypes are focused developments of some aspect of the system which may advance evolutionary change. Prototypes may be developed without anticipation of the resulting software being directly included in a formal release. Prototypes are developed on a faster time scale than the incremental and formal development track.

raw data	Data in their original packets, as received from the spacecraft and instruments, unprocessed by EDOS. o Level 0 – Raw instrument data at original resolution, time ordered, with duplicate packets removed. o Level 1A – Level 0 data, which may have been reformatted or transformed reversibly, located to a coordinate system, and packaged with needed ancillary and engineering data. o Level 1B – Radiometrically corrected and calibrated data in physical units at full instrument resolution as acquired. o Level 2 – Retrieved environmental variables (e.g., ocean wave height, soil moisture, ice concentration) at the same location and similar resolution as the Level 1 source data. o Level 3 – Data or retrieved environmental variables that have have been spatially and/or temporally resampled (i.e., derived from Level 1 or Level 2 data products). Such resampling may include averaging and compositing. o Level 4 – Model output and/or variables derived from lower level data which are not directly measured by the instruments. For example, new variables based upon a time series of Level 2 or Level 3 data.
real-time data	Data that are acquired and transmitted immediately to the ground (as opposed to playback data). Delay is limited to the actual time required to transmit the data.
reconfiguration	A change in operational hardware, software, data bases or procedures brought about by a change in a system's objectives.
SCC-stored commands and tables	Commands and tables which are stored in the memory of the central onboard computer on the spacecraft. The execution of these commands or the result of loading these operational tables occurs sometime following their storage. The term "core-stored" applies only to the location where the items are stored on the spacecraft and instruments; core-stored commands or tables could be associated with the spacecraft or any of the instruments.
scenario	A description of the operation of the system in user's terminology including a description of the output response for a given set of input stimuli. Scenarios are used to define operations concepts.